# 信创大赛实验操作指南

武汉达梦数据库有限公司内部资料

达梦数据库信创大赛资料

## 版权声明

版权所有©武汉达梦数据库有限公司。本指南所有内容受版权法保护，达梦拥有所有版权，但注明引用其他方的内容除外。未经武汉达梦数据库有限公司事先书面许可，任何人、任何组织不得将本书的任何内容以任何方式进行复制、抄袭、经销、翻印、存储于信息检索系统或使用于任何其他商业目的。

达梦数据库信创大赛资料

# 目 录

实验 1 LINUX 平台安装 DM8 企业版........................................................ 1
实验 2 卸载 DM8 企业版........................................................10
实验 3 创建数据库实例和运行 DM8........................................................13
实验 4 运行 DM8........................................................21
实验 5 删除数据库实例........................................................23
实验 6 用户和角色管理........................................................ 28
实验 7 简单查询........................................................32
实验 8 单行函数........................................................41
实验 9 分组函数........................................................45
实验 10 联合查询........................................................51
实验 11 子查询........................................................57
实验 12 DML 语句........................................................58
实验 13 表管理........................................................62
实验 14 约束管理........................................................67
实验 15 视图管理........................................................72
实验 16 索引管理........................................................76
实验 17 数据库开归档........................................................80
1、 命令方式开归档........................................................80
2、 利用管理工具开归档........................................................81
实验 18 物理备份与还原........................................................ 83
1、 数据库冷备........................................................83
2、 数据库热备........................................................86
3、数据库还原........................................................89



达梦数据库信创大赛资料

实验 19 逻辑备份与还原........................................................................................ 91

达梦数据库信创大赛资料

# 实验 1 LINUX 平台安装 DM8 企业版

## 实验目的：

熟悉在 LINUX 平台安装 DM8 企业版，并且能够排除安装过程中的错误。

## 实验要求：

1. LINUX（GLIBC 2.3 以上，内核 2.6，已安装 KDE/GNOME 桌面环境，建议预先安装 UNIXODBC 组件），**root** 登录。

2. 创建**DM**数据库系统用户

以下步骤只作为参考提示，具体步骤及操作请以本机系统为准，具体细节可向**LINUX**系统管理员咨询。

1) 创建安装用户组

groupadd dinstall

2) 创建安装用户

useradd -g dinstall -m -d /home/dmdba -s /bin/bash dmdba

3) 初始化用户密码

passwd dmdba 之后通过系统提示进行密码设置。

4) 创建达梦数据库的安装目录

mkdir /dm8

5) 对目录的创建者的更改

chown -R dmdba.dinstall /dm8

3. 修改操作系统资源限制

在LINUX、SOLARIS、AIX和HP-UNIX等系统中，因为ULIMIT命令的

达梦数据库信创大赛资料

存在，会对程序使用操作系统资源进行限制。为了使达梦数据库能够正常运行，建议用户检查ULIMIT的参数。用户可运行ulimit -a 进行查询。

1) DATA SEG SIZE (KBYTES, -D)

建议用户设置为1048576(即1GB)以上或UNLIMITED(无限制)，此参数过小将导致数据库启动失败。

2) FILE SIZE (BLOCKS, -F)

建议用户设置为UNLIMITED(无限制)。

3) OPEN FILES (-N)

建议用户设置为65536以上或UNLIMITED(无限制)。

或在同一个会话root账号执行ulimit -n 65536

4) VIRTUAL MEMORY (KBYTES, -V)

建议用户设置为 1048576(即 1GB)以上或 UNLIMITED(无限制)，此参数过小将导致数据库启动失败。

## 实验步骤：

DM 的安装向导在 LINUX 平台下与 WINDOWS 下非常相似。

安装过程如下：

1． 加载光驱以安装达梦数据库的用户登录到 LINUX 系统，将 DM 安装光盘放入光驱，然后加载（mount）光驱。

一般可以通过执行下面的命令来加载光驱：

mount -o loop /dev/cdrom /mnt 这里假定光驱对应的文件为/dev/cdrom 且目标路径/mnt 已存在。

2． 启动安装程序在 LINUX 的 KDE 或 GNOME 图形界面（可通过 startx 命令进入）下，点击光盘下的 DMInstall.bin 启动安装程序（或直接进入/mnt/cdrom 目录后执行下面的命令：./ DMInstall.bin），将出现 LINUX 下的 DM 安装界面。如图 1.1 所示。



达梦数据库信创大赛资料




图 1.1

说明：如果不能够运行，请检查当前用户对 DMInstall.bin 是否具有执行权限，如果没有则使用 chmod 命令增加当前用户的执行权限或直接向 LINUX 系统管理员咨询。安装前还应检查是否有足够的磁盘空间。

请根据系统配置选择相应语言与时区，默认为“简体中文”与“GTM+08：00 中国标准时间”，点击“确认”按钮继续安装。如图 1.2 所示。




图 1.2

安装欢迎界面，点击“开始”按钮继续安装。如图 1.3 所示。




图 1.3

在安装和使用 DM 之前，该安装程序需要用户阅读许可协议条款，用户如接受该协议，则选中“接受”，并点击“下一步”继续安装；用户若选中“不接受”，将无法进行安装。如图 1.4 所示。




图 1.4

验证 KEY 文件，用户点击“浏览”按钮， 选取 KEY 文件，安装程序将

自动验证 KEY 文件信息。如果是合法的 KEY 文件且在有效期内，用户可以点击“下一步”继续安装，如图 1.5 所示。

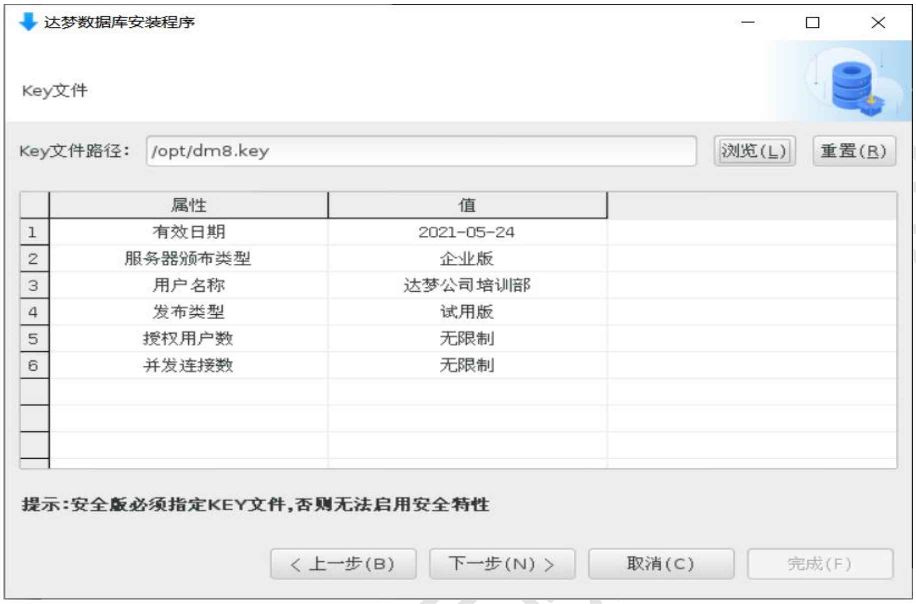



图 1.5

选择安装方式，DM 安装程序提供四种安装方式：“典型安装”、“服务器安装”、“客户端安装”和“自定义安装”，用户可根据实际情况灵活地选择，如图 1.6 所示。

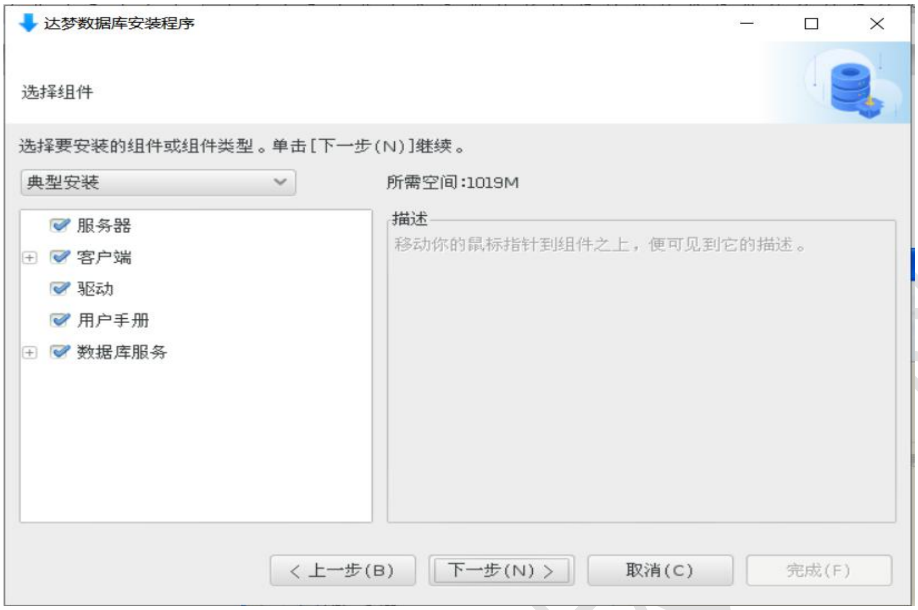



图 1.6

达梦数据库默认安装在/opt/dmdbms 目录下，用户可以通过点击“浏览”按钮自定义安装目录。安装路径里的目录名由英文字母、数字和下划线等组成，不支持包含空格的目录名，建议不要使用中文字符等。如图 1.7 所示。



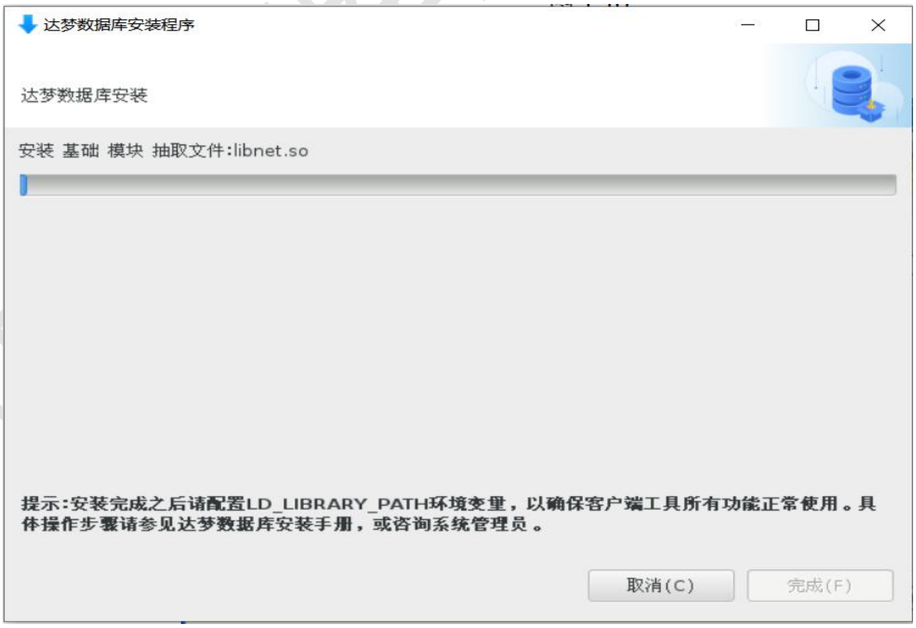

图 1.7

安装前小结，显示用户即将进行的安装的有关信息，例如产品名称、版本信息、安装类型、安装目录、是否初始化数据库、可用空间、可用内存等信息，用户检查无误后点击“安装”按钮，开始拷贝安装的软件。如图 1.8 所示。




图 1.8

点击安装。如图 1.9 所示。



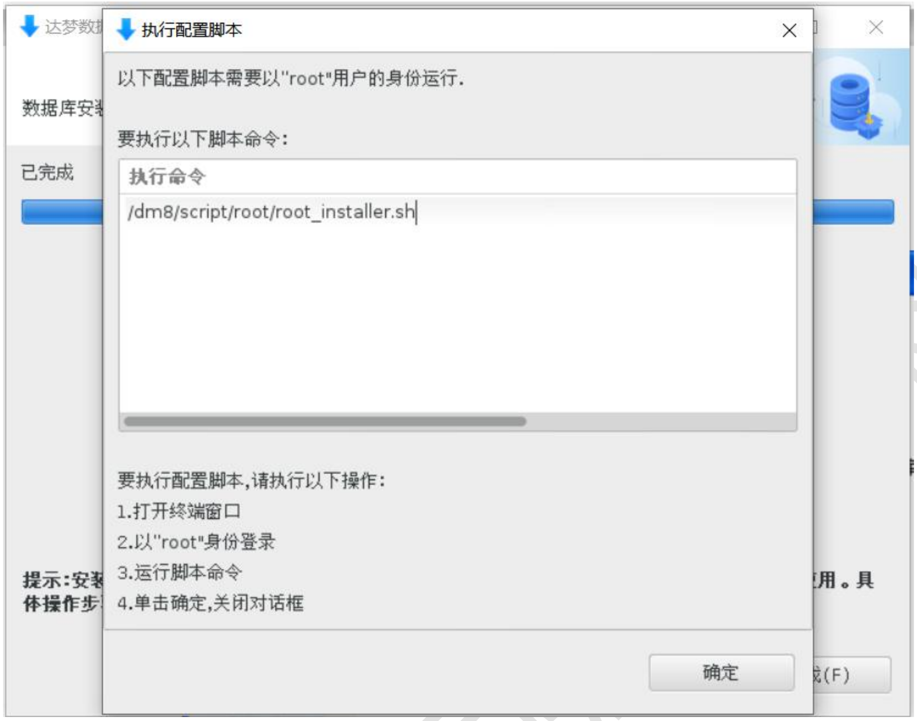






图 1.9

点击完成，完成数据库软件的安装，如图 1.10 所示。

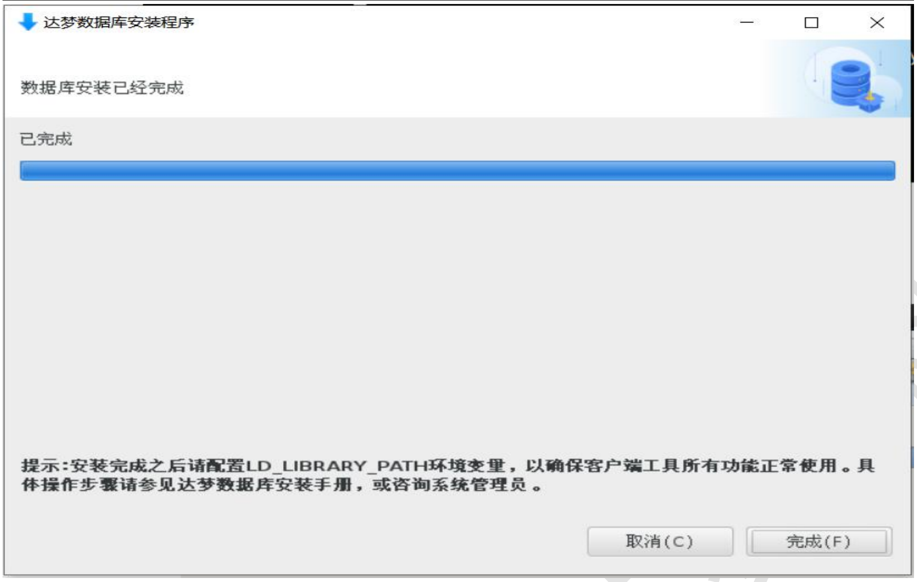



图 1.10

3. 卸载光驱安装完后一般要卸载光驱才能取出光盘。与前面的 mount 光驱命令相对应，可以使用下面的命令来卸载光驱：umount /dev/cdrom

4. 安装完成之后，将提示配置环境变量。用户手动配置环境变量，将安装目录的 bin 文件夹路径添加到环境变量，具体操作步骤请咨询 LINUX 管理员。

```
[dmdba@dca01 ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

export DM_HOME=/dm7
export PATH=$DM_HOME/bin:$DM_HOME/tool:$PATH:$HOME/bin
[dmdba@dca01 ~]$
```

# 实验 2 卸载 DM8 企业版

## 实验目的：

掌握卸载 DM8 企业版。

## 实验步骤：

LINUX 操作系统中执行卸载快捷方式，或者执行安装目录下的 uninstall.sh 脚本卸载。程序将会弹出提示框确认是否卸载程序。如图 3.1 所示。

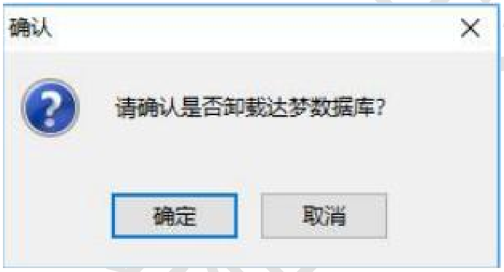



图 2.1

点击确认，进入卸载程序。如图 2.2

达梦数据库信创大赛资料




图 2.2

点击“卸载”按钮，卸载程序便会运行，弹出正在卸载对话框，如图 2.3 所示.




图 2.3

卸载完成窗口如图 2.4 所示，点击“完成”按钮，结束卸载程序。卸载程序不会删除安装目录下有用户数据的库文件以及安装 DM 后使用过程中产生的一些文件。用户可以根据需要手工删除这些内容。




图 2.4

# 实验3 创建数据库实例和运行DM8

## 实验目的：

掌握创建 DM 的数据库实例

## 实验步骤：

数据库配置工具的图形界面在各个操作系统上保持一致，用户可通过数据库配置工具来设置数据库初始化的参数及属性，并且选择是否安装示例库。

步骤 1：选择操作方式，用户可选择创建数据库实例、删除数据库实例、注册数据库服务和删除数据库服务四种操作方式，本章只详细介绍创建数据库实例的使用步骤，删除数据库实例、注册数据库服务和删除数据库服务的详细操作，请参见达梦数据库联机帮助。如图 3.1 所示。




图 3.1

选择创建数据库实例，点击“开始”确认。

步骤 **2**：创建数据库模板，系统提供三套数据库模板供用户选择：一般用途、联机分析处理和联机事务处理，用户可根据自身的用途选择相应的模板。如图 3.2 所示。




图 3.2

步骤 **3**：选择数据库目录，用户可通过浏览或是输入的方式选择数据库所在目录，如 3.3 所示。

达梦数据库

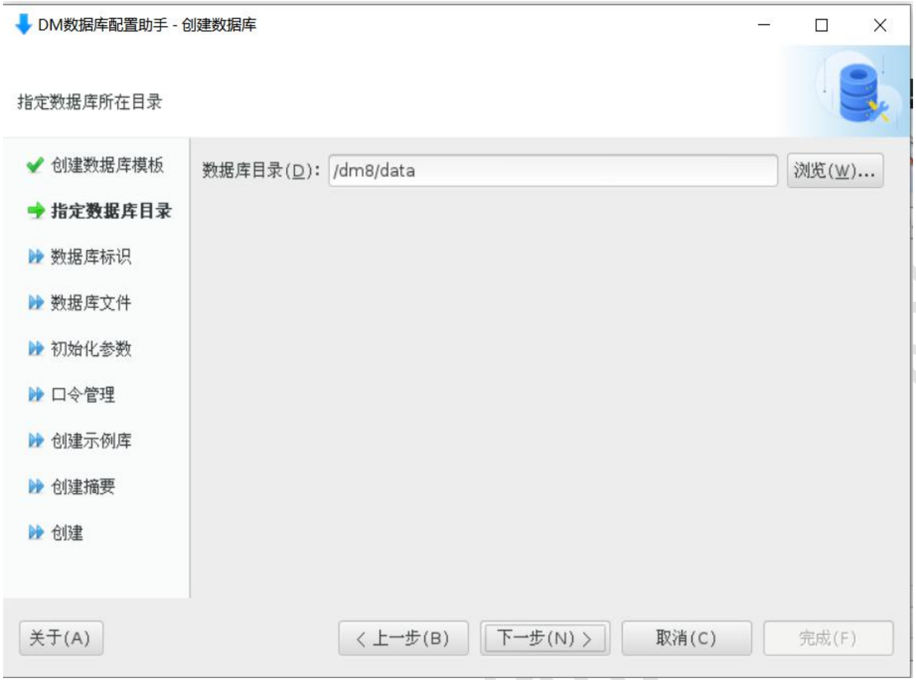



图 3.3

步骤 **4**：输入数据库标识，用户可输入数据库名称、实例名、端口号等参数，如图 3.4 所示。




图 3.4

步骤 **5**：数据数据库文件所在位置，用户可通过选择或输入确定数据库控制文件、数据库日志文件的所在位置，并可通过右侧功能按钮，对文件进行添加或删除，如图 3.5。

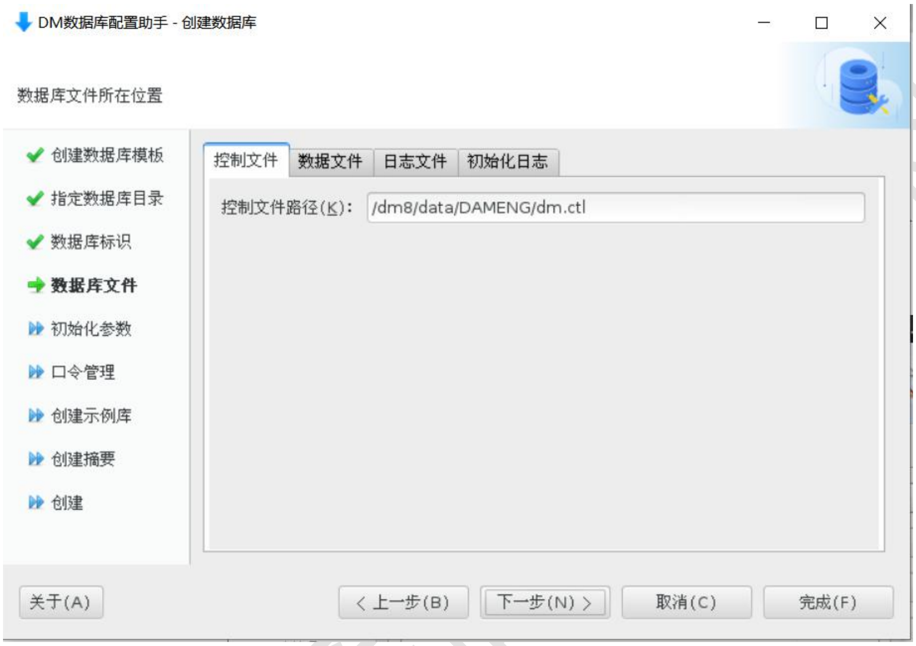



图 3.5

步骤 **6**：数据库初始化参数，用户可输入数据库相关参数，如簇大小、页大小、日志文件大小、是否大小写敏感是否使用 UNICODE 等，如图 3.6。




图 3.6

步骤 **7**：口令管理，用户可输入 SYSDBA，SYSAUDITOR 的密码，对默认口令进行更改，如果安装版本为安全版，将会增加 SYSSSO 用户的密码修改，如图 3.7。




图 3.7

步骤 **8**：选择创建示例库，用户可选择是否创建示例库。如图 3.8




图 3.8

步骤 **9**：创建数据库摘要，在安装数据库之前，将显示用户通过数据库配置工具设置的相关参数，如图 3.9.

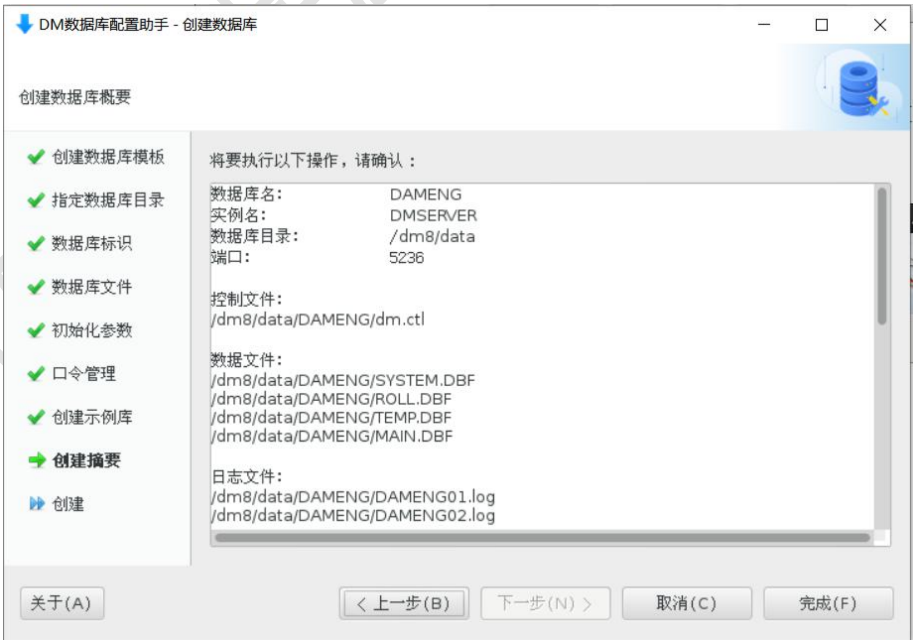


图 3.9

步骤 **10**：安装初始化数据库，如图 3.10 所示

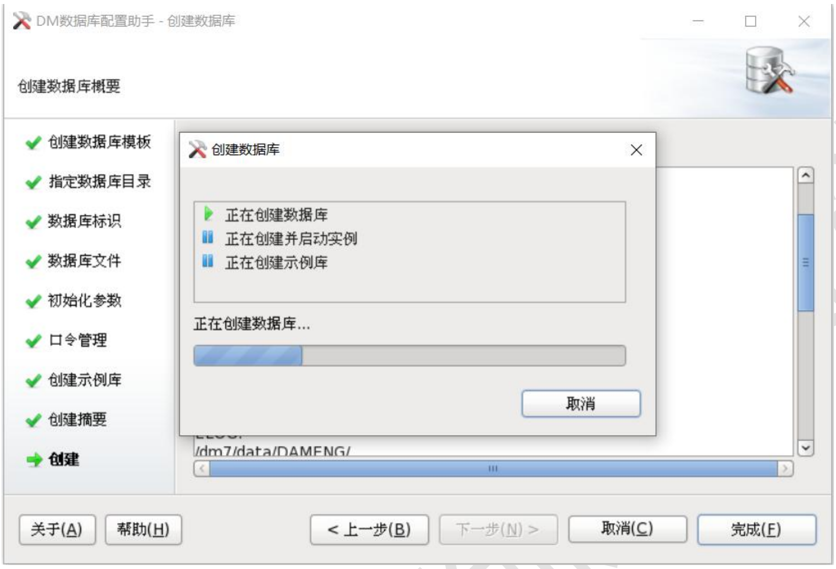



图 3.10

安装完成后将弹出数据库相关参数及文件位置，如图 3.11。

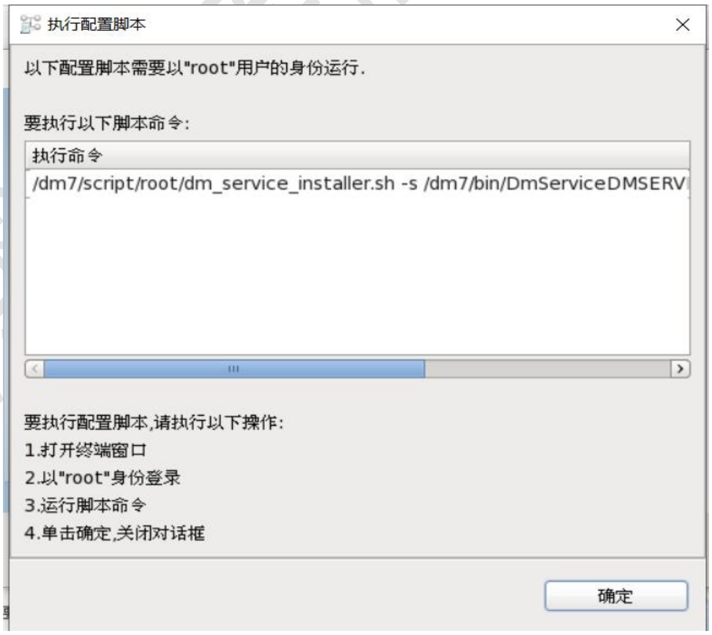


图 3.11

[root@local ~]# /dm8/script/root/dm_service_installer.sh -s /dm8/bin/DmServiceDMSERVER




达梦数据库信创大赛

# 实验 4 运行 DM8

## 实验目的：

启动与停止 DM8 服务器是数据库管理员（DBA）的重要职责。只有启动 DM 服务器后，客户端软件才能连接到数据库，并对其进行管理和数据操作。如果数据库服务器没有启动或数据库服务器被关闭，则用户无法连接和使用数据库。

## 实验步骤：

**LINUX(UNIX)**操作系统

DM安装成功后服务自动启动。在DM服务停止的情况下，可以采取以下两种方式启动。第一种启动服务器的步骤为：

启动终端，执行以下命令，进入DM服务器程序安装的目录：

(注：假设 DM 安装目录为/dm8/)

cd /dm8/bin

然后拷贝将要启动数据库的 dm.ini 文件到当前目录，执行命令：

./dmserver 或者在 dmserver 后面，添加 ini 文件路径，假设 ini 文件路径为 /dm8/data/DAMENG/dm.ini ，执行命令： ./dmserver /dm8/data/DAMENG/dm.in 当显示 SYSTEM IS READY 后，服务器已经完全启动成功，如图 4.1 所示。

达梦数据库信创大赛资料

```
[dmdba@localhost bin]$ ./dmserver /dm8/data/DAMENG/dm.ini
version info: enterprise
Use normal os_malloc instead of HugeTLB
Use normal os_malloc instead of HugeTLB
DM Database Server x64 V8 1-1-78-20.04.28-121039-ENT  startup...
Database mode = 0, oguid = 0
License will expire on 2021-05-24
file lsn: 55324
ndct db load finished
ndct fill fast pool finished
iid page's trxid[6015]
NEXT TRX ID = 6016
pseg_collect_items, collect 0 active_trxs, 0 cmt_trxs, 0 pre_cmt_trxs, 0 active_pages,
 0 cmt_pages, 0 pre_cmt_pages
pseg_process_collect_items end, 0 active trx, 0 active pages, 0 committed trx, 0 commi
tted pages
total 0 active crash trx, pseg_crash_trx_rollback begin ...
pseg_crash_trx_rollback end
purg2_crash_cmt_trx end, total 0 page purged
set EP[0]'s pseg state to inactive
pseg recv finished
nsvr_startup end.
aud sys init success.
aud rt sys init success.
systables desc init success.
ndct_db_load_info success.
nsvr_process_before_open begin.
nsvr_process_before_open success.
total 0 active crash trx, pseg_crash_trx_rollback begin ...
pseg_crash_trx_rollback end
SYSTEM IS READY.
```

图 4.1

当需要停止服务器时，输入“exit”命令，服务器将关闭当前所有的客户端连接，停止服务。当系统退回到操作系统提示符时，服务器就已经完全退出。

另外一种启动服务器的步骤为： 以启动系统服务的方式启动数据库。具体步骤及操作请以本机系统为准，具体细节可向 **LINUX** 系统管理员咨询。启动终端后，执行以下命令： （注：数据库服务名称为：DmService+实例名。实例名则是在数据库配置工具中配置得来，以实例名为 DMSERVER 为例 systemctl start DmServiceDMSERVER.serviceservice 当显示 starting DmServiceDMSERVER: [OK]后，服务器已经完全启动成功。

当需要停止服务器时，输入以下命令： systemctl stop DmServiceDMSERVER.serviceservice

当显示 stoping DmServiceDMSERVER: [OK]后，服务器将关闭当前所有的

客户端连接，停止服务。当系统退回到操作系统提示符时，服务器就已经完全退出。

3、通过图形化界面来启停服务

使用 root 账号执行：/dm8/tool/dmserver.sh




在图形界面上，直接停止或是启动服务。

# 实验 5 删除数据库实例

## 实验目的：

掌握删除 DM8 数据库实例。

## 实验要求：

在删除数据库实例之前，作为一名称职的 DBA，要把当前的数据库做一次完整的备份，备份操作详见实验 22。

在删除数据库之前要停止当前正在运行的数据库实例。

## 实验步骤：

步骤 1：停止当前正在运行的数据库实例。如图 5.1 所示。




图 5.1

步骤 2：启动 dbca.sh 工具，（/dm8/tool/dbca.sh)选择操作方式，用户可选择创建数据库实例、删除数据库实例、注册数据库服务和删除数据库服务四种操作方式，本章只详细介绍删除数据库实例，如图 5.2 所示。




图 5.2

步骤 3：点击“开始”，选择要删除的数据库实例名。如图 5.3 所示。




图 5.3

步骤 4：点击“下一步”，提示删除资源的信息。如图 5.4 所示。




图 5.4

步骤 5：点击“完成”，提示确认删除数据库实例信息，如图 5.5 所示。




图 5.5

步骤 6：点击“确认”，完成数据库实例删除操作，如图 5.6 所示。




图 5.6

达梦数据库信创大赛资料

# 实验 6 用户和角色管理

## 实验目的：

用户管理、权限管理、角色管理，这些内容属于高级管理范畴，是数据库管理员应该掌握的内容。

**语法：**

CREATE USER 用户名 IDENTIFIED BY "口令" PASSWORD POLICY 口令策略 ENCRYPT BY <口令> DISKSPACE LIMIT 空间限制[NOT] READ ONLY LIMIT 资源限制[NOT] ALLOW IP 地址[NOT] ALLOW DATETIME 时间 DEFAULT TABLESPACE <表空间名>;

## 实验步骤：

为当前数据库创建一个名为 TEST 的用户：

```
CREATE USER TEST IDENTIFIED BY TESTTEST123;
```

查看创建用户的信息：

```
SELECT USER_NAME,USER_ID FROM DBA_USERS WHERE USER_NAME=' TEST' ;
```

结果如下：

| USER_NAME | USER_ID |
|---|---|
| TEST | 50331748 |

修改用户“TEST”的 RESOURCE 的信息，如果用户失败的登录次数达到 3 次，这个用户帐号将被锁定。

```
ALTER USER TEST LIMIT FAILED_LOGIN_ATTEMPS 3;
```

查看用户资源信息：

```
SELECT ID,FAILED_NUM FROM SYSUSERS WHERE ID='50331748';
```

结果如下：

| ID | FAILED_NUM |
| --- | --- |
| 50331748 | 3 |

给用户分配系统权限，让“TEST”用户具有创建表的权限：

```
GRANT CREATE TABLE TO TEST;
```

查看用户权限信息：

```
SELECT GRANTEE,PRIVILEGE FROM DBA_SYS_PRIVS WHERE GRANTEE=’TEST’;
```

结果如下：

| GRANTEE | PRIVILEGE |
| --- | --- |
| TSET | CREATE TABLE |

给用户分配对象权限，让“TEST”用户具有查看 SYSDBA.EM 权限：

```
GRANT SELECT ON SYSDBA.EMP TO TEST;
```

查看用户权限信息：

```
SELECT GRANTEE,PRIVILEGE,TABLE_NAME FROM DBA_TAB_PRIVS WHERE
GRANTEE=’TEST’;
```

结果如下：

| GRANTEE | PRIVILEGE | TABLE_NAME |
| --- | --- | --- |
| TSET | SELECT | EMP |

注意：系统权限和对象权限在分配给用户时，不能写在一条 SQL 中。

撤销用户的权限，撤销“TEST”创建表的权限：

```
REVOKE CREATE TABLE FROM TEST;
```

查看用户权限信息：

```
SELECT GRANTEE,PRIVILEGE FROM DBA_SYS_PRIVS WHERE GRANTEE=’TSET’;
```

结果如下：

| GRANTEE | PRIVILEGE |
|---|---|

给用户分配对象权限,让“TEST”用户具有查看 SYSDBA.EM 权限：

```
REVOKE SELECT ON SYSDBA.EMP FROM TEST;
```

查看用户权限信息：

```
SELECT GRANTEE,PRIVILEGE,TABLE_NAME FROM DBA_TAB_PRIVS WHERE

GRANTEE=’ TEST’ ;
```

结果如下：

| GRANTEE | PRIVILEGE | TABLE_NAME |
|---|---|---|

删除用户”TEST”：

```
DROP USER TEST;
```

角色是一组权限的组合，使用角色的目的是使权限管理更加方便。

为数据库创建一个名为 ROLE1 的角色,并且分配创建表和跟新 SYSDBA.EMP 表的权限：

```
CREATE ROLE ROLE1;
```

查看角色“ROLE1”的信息：

```
SELECT ROLE FROM DBA_ROLES WHERE ROLE=’ ROLE1’ ;
```

结果如下：

| GRANTEE |
|---|
| ROLE1 |

给“ROLE1”分配权限：

```
GRANT CREATE TABLE TO ROLE1;

GRANT UPDATE ON SYSDBA.EMP TO ROLE1;
```

把”ROLE1”分配给”TSET”用户:

```
GRANT ROLE1 TO TEST;
```

把”ROLE1”的角色从”TEST”用户撤销:

```
REVOKE ROLE1 FROM TEST;
```

禁用角色”ROLE1”：

```
CALL SP_SET_ROLE(‘ROLE1’,0);
```

启用角色”ROLE1”

```
CALL SP_SET_ROLE(‘ROLE1’,1);
```

删除角色”ROLE1”：

```
DROP ROLE ROLE1;
```

达梦数据库信创

# 实验 7 简单查询

## 实验目的：

从指定表中取出指定的列的数据

## 实验步骤：

**语法：**

从数据库中选取资料列，并允许从一或多个资料表中，选取一或多个资料列或资料行。SELECT 陈述式的完整语法相当复杂，但主要子句可摘要为：

```
SELECT SELECT_LIST
[ INTO NEW_TABLE ]
FROM TABLE_SOURCE
[ WHERE SEARCH_CONDITION ]
[ GROUP BY GROUP_BY_EXPRESSION ]
[ HAVING SEARCH_CONDITION ]
[ ORDER BY ORDER_EXPRESSION [ ASC | DESC ] ]
```

**例：**

“PERSONS” 表中的数据有

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
| --- | --- | --- | --- |
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES |
| SVENDSON | TOVE | BORGVN 23 | SANDNES |
| PETTERSEN | KARI | STORGT 20 | STAVANGER |

选出字段名“LASTNAME”、“ FIRSTNAME” 的数据

```
SELECT LASTNAME,FIRSTNAME FROM PERSONS;
```

返回结果:

| LASTNAME | FIRSTNAME |
| --- | --- |



达梦数据库信创大赛资料

| | |
|---|---|
| HANSEN | OLA |
| SVENDSON | TOVE |
| PETTERSEN | KARI |

选出所有字段的数据

```
SELECT * FROM PERSONS;
```

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES |
| SVENDSON | TOVE | BORGVN 23 | SANDNES |
| PETTERSEN | KARI | STORGT 20 | STAVANGER |

ALIAS

**用途：**

可用在表、结果集或者列上，为它们取一个逻辑名称。

**语法：**

给列取别名:

```
SELECT COLUMN AS COLUMN_ALIAS FROM TABLE;
```

给表取别名:

```
SELECT COLUMN FROM TABLE AS TABLE_ALIAS;
```

**例：**

“PERSONS”表中的原始数据:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES |
| SVENDSON | TOVE | BORGVN 23 | SANDNES |

| | | | |
|---|---|---|---|
| PETTERSEN | KARI | STORGT 20 | STAVANGER |

运行下面的 SQL：

```
SELECT LASTNAME AS FAMILY, FIRSTNAME AS NAME FROM PERSONS;
```

返回结果：

| FAMILY | NAME |
|---|---|
| HANSEN | OLA |
| SVENDSON | TOVE |
| PETTERSEN | KARI |

运行下面的 SQL：

```
SELECT LASTNAME, FIRSTNAME FROM PERSONS AS EMPLOYEES;
```

返回结果：

EMPLOYEES 中的数据有：

| LASTNAME | FIRSTNAME |
|---|---|
| HANSEN | OLA |
| SVENDSON | TOVE |
| PETTERSEN | KARI |

**WHERE**

**用途：**

被用来规定一种选择查询的标准

**语法：**

```
SELECT COLUMN FROM TABLE WHERE COLUMN CONDITION VALUE;
```

下面的操作符能被使用在 WHERE 中：

=,<>,>,<,>=,<=,BETWEEN,LIKE

注意： 在某些 SQL 的版本中不等号< >能被写作为!=



达梦数据库信创大赛资料

**解释：**

SELECT 语句返回 WHERE 子句中条件为 TRUE 的数据

**例：**

从“PERSONS”表中选出生活在“SANDNES” 的人

```
SELECT * FROM PERSONS WHERE CITY='SANDNES';
```

“PERSONS”表中的数据有:

| LASTNAME | FIRSTNAME | ADDRESS | CITY | YEAR |
|---|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES | 1951 |
| SVENDSON | TOVE | BORGVN 23 | SANDNES | 1978 |
| SVENDSON | STALE | KAIVN 18 | SANDNES | 1980 |
| PETTERSEN | KARI | STORGT 20 | STAVANGER | 1960 |

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY | YEAR |
|---|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES | 1951 |
| SVENDSON | TOVE | BORGVN 23 | SANDNES | 1978 |
| SVENDSON | STALE | KAIVN 18 | SANDNES | 1980 |

AND & OR

**用途：**

在 WHERE 子句中 AND 和 OR 被用来连接两个或者更多的条件

**解释：**

AND 在结合两个布尔表达式时，只有在两个表达式都为 TRUE 时才传回 TRUE

OR 在结合两个布尔表达式时，只要其中一个条件为 TRUE 时，OR 便传回 TRUE

**例：**

“PERSONS”表中的原始数据:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES |
| SVENDSON | TOVE | BORGVN 23 | SANDNES |
| SVENDSON | STEPHEN | KAIVN 18 | SANDNES |

用 AND 运算子来查找“PERSONS” 表中 FIRSTNAME 为“TOVE”而且 LASTNAME 为“SVENDSON”的数据

```
SELECT * FROM PERSONS
WHERE FIRSTNAME='TOVE'
AND LASTNAME='SVENDSON';
```

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| SVENDSON | TOVE | BORGVN 23 | SANDNES |

用 OR 运算子来查找“PERSONS” 表中 FIRSTNAME 为“TOVE”或者 LASTNAME 为“SVENDSON”的数据

```
SELECT * FROM PERSONS
WHERE FIRSTNAME='TOVE'
OR LASTNAME='SVENDSON';
```

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| SVENDSON | TOVE | BORGVN 23 | SANDNES |
| SVENDSON | STEPHEN | KAIVN 18 | SANDNES |

你也能结合 AND 和 OR (使用括号形成复杂的表达式),如:

```
SELECT * FROM PERSONS
```

```
WHERE (FIRSTNAME='TOVE' OR FIRSTNAME='STEPHEN')
AND LASTNAME='SVENDSON';
```

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| SVENDSON | TOVE | BORGVN 23 | SANDNES |
| SVENDSON | STEPHEN | KAIVN 18 | SANDNES |

**BETWEEN…AND**

**用途：**

指定需返回数据的范围

**语法：**

```
SELECT COLUMN_NAME FROM TABLE_NAME
WHERE COLUMN_NAME
BETWEEN VALUE1 AND VALUE2;
```

**例：**

“PERSONS”表中的原始数据

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES |
| NORDMANN | ANNA | NESET 18 | SANDNES |
| PETTERSEN | KARI | STORGT 20 | STAVANGER |
| SVENDSON | TOVE | BORGVN 23 | SANDNES |

用 BETWEEN…AND 返回 LASTNAME 为从“HANSEN”到“PETTERSEN”的数据：

```
SELECT * FROM PERSONS WHERE LASTNAME
```

```
BETWEEN 'HANSEN' AND 'PETTERSEN';
```

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| HANSEN | OLA | TIMOTEIVN 10 | SANDNES |
| NORDMANN | ANNA | NESET 18 | SANDNES |
| PETTERSEN | KARI | STORGT 20 | STAVANGER |

为了显示指定范围之外的数据，也可以用 NOT 操作符:

```
SELECT * FROM PERSONS WHERE LASTNAME
NOT BETWEEN 'HANSEN' AND 'PETTERSEN';
```

返回结果:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| SVENDSON | TOVE | BORGVN 23 | SANDNES |

**DISTINCT**

**用途：**

DISTINCT 关键字被用作返回唯一的值

**语法：**

```
SELECT DISTINCT COLUMN-NAME(S) FROM TABLE-NAME;
```

**解释：**

当 COLUMN-NAME(S)中存在重复的值时，返回结果仅留下一个

**例：**

“ORDERS”表中的原始数据

| COMPANY | ORDERNUMBER |
|---|---|
| SEGA | 3412 |

| | |
|---|---|
| W3SCHOOLS | 2312 |
| TRIO | 4678 |
| W3SCHOOLS | 6798 |

用 DISTINCT 关键字返回 COMPANY 字段中唯一的值：

```
SELECT DISTINCT COMPANY FROM ORDERS;
```

返回结果：

| COMPANY |
|---|
| SEGA |
| W3SCHOOLS |
| TRIO |

ORDER BY

**用途：**

指定结果集的排序

**语法：**

```
SELECT COLUMN-NAME(S) FROM TABLE-NAME ORDER BY { ORDER_BY_EXPRESSION
[ ASC | DESC ] }
```

**解释：**

指定结果集的排序，可以按照 ASC(递增方式排序，从最低值到最高值)或者 DESC(递减方式排序，从最高值到最低值)的方式进行排序，默认的方式是 ASC

**例：**

“ORDERS”表中的原始数据：

| COMPANY | ORDERNUMBER |
|---|---|
| SEGA | 3412 |



达梦数据库信创大赛资料

| | |
|---|---|
| ABC SHOP | 5678 |
| W3SCHOOLS | 2312 |
| W3SCHOOLS | 6798 |

按照 COMPANY 字段的升序方式返回结果集：

```
SELECT COMPANY, ORDERNUMBER FROM ORDERS
ORDER BY COMPANY;
```

返回结果：

| COMPANY | ORDERNUMBER |
|---|---|
| ABC SHOP | 5678 |
| SEGA | 3412 |
| W3SCHOOLS | 6798 |
| W3SCHOOLS | 2312 |

按照 COMPANY 字段的降序方式返回结果集：

```
SELECT COMPANY, ORDERNUMBER FROM ORDERS
ORDER BY COMPANY DESC;
```

返回结果：

| COMPANY | ORDERNUMBER |
|---|---|
| W3SCHOOLS | 6798 |
| W3SCHOOLS | 2312 |
| SEGA | 3412 |
| ABC SHOP | 5678 |

# 实验 8 单行函数

## 实验目的：

数值函数、日期时间函数、类型转换函数等使用。

## 实验步骤：

**数值函数：**

**用途：**

数值函数接受数值参数并返回数值作为结果

**语法：**

SELECT SINGLE- FUNCTION FROM TABLE_NAME;

**例：**

“ORDERS”表中的原始数据：

| COMPANY | ORDERNUMBER |
|---|---|
| SEGA | 3412.26 |

用 ROUND 操作，ROUND 操作是做四舍五入。

```
SELECT COMPANY,ROUND(ORDERNUMBER,1) FROM ORDERS;
```

返回的结果为：

| COMPANY | ROUND(ORDERNUMBER,1) |
|---|---|
| SEGA | 3412.3 |

用 TRUNC 操作，TRUNC 是做截断操作。

```
SELECT COMPANY,TRUNC(ORDERNUMBER,1) FROM ORDERS;
```

返回结果为：

| COMPANY | TRUNC(ORDERNUMBER,1) |
|---|---|
| SEGA | 3412.2 |

达梦数据库信创大赛资料

**日期时间函数:**

**用途:**

日期时间函数的参数至少有一个是日期时间类型(TIME，DATE，TIMESTAMP)，返回值一般为日期时间类型和数值类型

**例:**

"SALES"表的原数据

| NAME | SALES | DATE1 |
|---|---|---|
| HANSEN | 1500 | 2011-09-02 |
| NORDMANN | 800 | 2012-08-10 |
| PETTERSEN | 1200 | 2013-07-12 |
| SVENDSON | 3000 | 2014-02-11 |

用 MONTHS_BETWEEN 操作，确定两个日期相隔的月数。

```
SELECT MONTHS_BETWEEN('2012-08-10','2011-09-02') AS D1 FROM SALES;
```

返回结果:

| D1 |
|---|
| 11.258064516129032 |

用 LAST_DAY 操作，月份的最后的一天

```
SELECT LAST_DAY(DATE1) AS D1 FROM SALES WHERE NAME='SVENDSON';
```

返回结果:

| D1 |
|---|
| 2014-02-28 |

用 NEXT_DAY，指定日期的下一天

```
SELECT NEXT_DAY(DATE1,’ 星期二’ ) AS D1 FROM SALES WHERE
NAME=’ SVENDSON’ ;
```

返回结果：

| D1 |
| --- |
| 2014-02-18 |

**类型转换函数:**

**用途:**

用转换构造函数可以将一个指定类型的数据转换为类的对象

语法:SELECT CONVERSION FUNCTION FROM TABLE_NAME;

**例:**

“SALES”表的原数据

| NAME | SALES | DATE1 |
| --- | --- | --- |
| HANSEN | 1500 | 2011-09-02 |
| NORDMANN | 800 | 2012-08-10 |
| PETTERSEN | 1200 | 2013-07-12 |
| SVENDSON | 3000 | 2014-02-11 |

用 TO_CHAR 操作，TO_CHAR 函数功能，就是将数值型或者日期型转化为字符型

**语法：**

SELECT TO_CHAR(COLUMN_NAME,FORMAT) FROM TABLE_NAME;

```
SELECT NAME,TO_CHAR(DATE1,’ YYYY-MM-DD’ ) AS D1 FROM SALES;
```

返回结果：

| NAME | D1 |
| --- | --- |
| HANSEN | 2011-09-02 |



达梦资料信创大赛资料

| | |
|---|---|
| NORDMANN | 2012-08-10 |
| PETTERSEN | 2013-07-12 |
| SVENDSON | 2014-02-11 |

用 TO_DATA 操作，TO_DATE 函数功能，就是把数值型和字符型转化为日期型

**语法：**

SELECT TO_DATE(COLUMN_NAME,FORMAT) FROM TABLE_NAME;

```
SELECT TO_DATE(‘201403’,’YYYY-MM’) AS D1 FROM DUAL;
```

返回结果：

| D1 |
|---|
| 2014-03-01 00:00:00.000000 |

注意：要熟悉当前数据库的时间格式的设置。上个例子在没有出现几号的时候，数据库默认为 1 号。

达梦数据库信创大赛资料

# 实验 9 分组函数

## 实验目的：

为了进一步方便用户的使用，增强查询能力，SQL 语言提供了多种内部集函数。集函数又称库函数，当根据某一限制条件从表中导出一组行集时，使用集函数可对该行集作统计操作。

## 实验步骤：

**聚集函数**

COUNT

**用途：**

传回选取的结果集中行的数目。

**语法：**

```
SELECT COUNT(COLUMN_NAME) FROM TABLE_NAME;
```

**例：**

“PERSONS”表中原始数据如下：

| NAME | AGE |
|---|---|
| HANSEN, OLA | 34 |
| SVENDSON, TOVE | 45 |
| PETTERSEN, KARI | 19 |

选取记录总数：

```
SELECT COUNT(NAME) FROM PERSONS;
```

执行结果：

| |
|---|
| 3 |

SUM



达梦数据库信创大赛资料

**用途：**

以表达式传回所有值的总和，或仅 DISTINCT 值。SUM 仅可用于数值资料行。已忽略 NULL 值。

**语法：**

```
SELECT SUM(COLUMN_NAME) FROM TABLE_NAME;
```

**例：**

“PERSONS”表中原始数据如下：

| NAME | AGE |
|---|---|
| HANSEN, OLA | 34 |
| SVENDSON, TOVE | 45 |
| PETTERSEN, KARI | 19 |

选取”PERSONS”表中所有人的年龄总和：

```
SELECT SUM(AGE) FROM PERSONS;
```

执行结果：

```
98
```

选取”PERSONS”表中年龄超过 20 岁的人的年龄总和：

```
SELECT SUM(AGE) FROM PERSONS WHERE AGE>20;
```

执行结果：

```
79
```

AVG

**用途：**

传回选取的结果集中值的平均值。已忽略 NULL 值。

**语法：**

```
SELECT AVG(COLUMN_NAME) FROM TABLE_NAME;
```

**例：**

“PERSONS”表中原始数据如下：

| NAME | AGE |
|---|---|
| HANSEN, OLA | 34 |
| SVENDSON, TOVE | 45 |
| PETTERSEN, KARI | 19 |

选取”PERSONS”表中所有人的平均年龄：

```
SELECT AVG(AGE) FROM PERSONS;
```

执行结果：

```
32.67
```

选取”PERSONS”表中年龄超过 20 岁的人的平均年龄：

```
SELECT AVG(AGE) FROM PERSONS WHERE AGE>20;
```

执行结果：

```
39.5
```

**MAX**

**用途：**

传回选取的结果集中值的最大值。已忽略 NULL 值。

**语法：**

```
SELECT MAX(COLUMN_NAME) FROM TABLE_NAME;
```

**例：**

“PERSONS”表中原始数据如下：

| NAME | AGE |
|---|---|
| HANSEN, OLA | 34 |

达梦数据库信创大赛设计

| | |
|---|---|
| SVENDSON, TOVE | 45 |
| PETTERSEN, KARI | 19 |

选取”PERSONS”表中的最大年龄：

```
SELECT MAX(AGE) FROM PERSONS;
```

执行结果：

| |
|---|
| 45 |

MIN

**用途：**

传回选取的结果集中值的最小值。已忽略 NULL 值。

**语法：**

```
SELECT MIN(COLUMN_NAME) FROM TABLE_NAME;
```

**例：**

“PERSONS”表中原始数据如下：

| NAME | AGE |
|---|---|
| HANSEN, OLA | 34 |
| SVENDSON, TOVE | 45 |
| PETTERSEN, KARI | 19 |

选取”PERSONS”表中的最小年龄：

```
SELECT MIN(AGE) FROM PERSONS;
```

执行结果：

| |
|---|
| 19 |

GROUP BY

**用途：**

对结果集进行分组，常与汇总函数一起使用。

**语法：**

```
SELECT COLUMN,SUM(COLUMN) FROM TABLE GROUP BY COLUMN;
```

**例：**

“SALES”表中的原始数据:

| COMPANY | AMOUNT |
|---|---|
| W3SCHOOLS | 5500 |
| IBM | 4500 |
| W3SCHOOLS | 7100 |

按照 COMPANY 字段进行分组，求出每个 COMPANY 的 AMOUT 的合计：

```
SELECT COMPANY,SUM(AMOUNT) FROM SALES
GROUP BY COMPANY;
```

返回结果：

| COMPANY | SUM(AMOUNT) |
|---|---|
| W3SCHOOLS | 12600 |
| IBM | 4500 |

HAVING

**用途：**

指定群组或汇总的搜寻条件。

**语法：**

```
SELECT COLUMN,SUM(COLUMN) FROM TABLE
GROUP BY COLUMN
HAVING SUM(COLUMN) CONDITION VALUE;
```

**解释：**

HAVING 通常与 GROUP BY 子句同时使用。不使用 GROUP BY 时，HAVING 则与 WHERE 子句功能相似。

**例：**

“SALES”表中的原始数据:

| COMPANY | AMOUNT |
|---|---|
| W3SCHOOLS | 5500 |
| IBM | 4500 |
| W3SCHOOLS | 7100 |

按照 COMPANY 字段进行分组，求出每个 COMPANY 的 AMOUT 的合计在 10000 以上的数据：

```
SELECT COMPANY,SUM(AMOUNT) FROM SALES
GROUP BY COMPANY HAVING SUM(AMOUNT)>10000;
```

返回结果：

| COMPANY | SUM(AMOUNT) |
|---|---|
| W3SCHOOLS | 12600 |

达梦数据库信创大赛组件

# 实验 10 联合查询

## 实验目的：

在关系数据库管理系统中，表建立时各数据之间的关系不必确定，常把一个实体的所有信息存放在一个表中。当检索数据时，通过连接操作查询出存放在多个表中的不同实体的信息。

## 实验步骤：

JOIN

**用途：**

当你要从两个或者以上的表中选取结果集时，你就会用到 JOIN。

**例：**

“EMPLOYEES”表中的数据如下，（其中 ID 为主键）：

| ID | NAME |
|---|---|
| 01 | HANSEN, OLA |
| 02 | SVENDSON, TOVE |
| 03 | SVENDSON, STEPHEN |
| 04 | PETTERSEN, KARI |

“ORDERS”表中的数据如下：

| ID | PRODUCT |
|---|---|
| 01 | PRINTER |
| 03 | TABLE |
| 03 | CHAIR |

用自然连接取数据：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT
```



达梦 信创大赛资料

```
FROM EMPLOYEES NATURALJOIN ORDERS;
```

返回结果：

| NAME | PRODUCT |
| --- | --- |
| HANSEN, OLA | PRINTER |
| SVENDSON, STEPHEN | TABLE |
| SVENDSON, STEPHEN | CHAIR |

注意：如果两张表有多列相同，不能用自然连接。

用 USING 取数据：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT
FROM EMPLOYEES JOIN ORDERS USING(ID);
```

返回结果：

| NAME | PRODUCT |
| --- | --- |
| HANSEN, OLA | PRINTER |
| SVENDSON, STEPHEN | TABLE |
| SVENDSON, STEPHEN | CHAIR |

注意：USING 内的字段不能有前缀。

用 EMPLOYEES 的 ID 和 ORDERS 的 ID 相关联选取数据：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT
FROM EMPLOYEES, ORDERS
WHERE EMPLOYEES.ID = ORDERS.ID;
```

达梦信创大赛资料

返回结果：

| NAME | PRODUCT |
| --- | --- |
| HANSEN, OLA | PRINTER |
| SVENDSON, STEPHEN | TABLE |
| SVENDSON, STEPHEN | CHAIR |

或者你也可以用 JOIN 关键字来完成上面的操作：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT
FROM EMPLOYEES
INNER JOIN ORDERS
ON EMPLOYEES.ID = ORDERS.ID;
```

**INNER JOIN 的语法：**

```
SELECT FIELD1, FIELD2, FIELD3
FROM FIRST_TABLE
INNER JOIN SECOND_TABLE
ON FIRST_TABLE.KEYFIELD = SECOND_TABLE.FOREIGN_KEYFIELD;
```

**解释：**

INNER JOIN 返回的结果集是两个表中所有相匹配的数据。

**LEFT JOIN 的语法：**

```
SELECT FIELD1, FIELD2, FIELD3
FROM FIRST_TABLE
LEFT JOIN SECOND_TABLE
ON FIRST_TABLE.KEYFIELD = SECOND_TABLE.FOREIGN_KEYFIELD;
```

用“EMPLOYEES”表去左外联结“ORDERS”表去找出相关数据：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT

FROM EMPLOYEES;

LEFT JOIN ORDERS

ON EMPLOYEES.ID = ORDERS.ID;
```

返回结果：

| NAME | PRODUCT |
|---|---|
| HANSEN, OLA | PRINTER |
| SVENDSON, TOVE | |
| SVENDSON, STEPHEN | TABLE |
| SVENDSON, STEPHEN | CHAIR |
| PETTERSEN, KARI | |

**解释：**

LEFT JOIN 返回“FIRST_TABLE”中所有的行尽管在“SECOND_TABLE”中没有相匹配的数据。

**RIGHT JOIN 的语法：**

```
SELECT FIELD1, FIELD2, FIELD3

FROM FIRST_TABLE

RIGHT JOIN SECOND_TABLE

ON FIRST_TABLE.KEYFIELD = SECOND_TABLE.FOREIGN_KEYFIELD;
```

用“EMPLOYEES”表去右外联结“ORDERS”表去找出相关数据：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT
```

达梦数据库信创大赛资料

```
FROM EMPLOYEES
RIGHT JOIN ORDERS
ON EMPLOYEES.ID = ORDERS.ID;
```

返回结果：

| NAME | PRODUCT |
|---|---|
| HANSEN, OLA | PRINTER |
| SVENDSON, STEPHEN | TABLE |
| SVENDSON, STEPHEN | CHAIR |

**解释：**

RIGHT JOIN 返回“SECOND_TABLE”中所有的行尽管在“FIRST_TABLE”中没有相匹配的数据。

**HASH JOIN 语法：**

```
SELECT FIELD1, FIELD2, FIELD3
FROM FIRST_TABLE
HASH JOIN SECOND_TABLE
ON FIRST_TABLE.KEYFIELD = SECOND_TABLE.FOREIGN_KEYFIELD;
```

用 HASH JOIN 取数据：

```
SELECT EMPLOYEES.NAME, ORDERS.PRODUCT
FROM EMPLOYEES HASH JOIN ORDERS
ON EMPLOYEES.ID = ORDERS.ID;
```

返回结果：

| NAME | PRODUCT |
|---|---|

达梦数据库信创大赛资料

| | |
|---|---|
| HANSEN, OLA | PRINTER |
| SVENDSON, STEPHEN | TABLE |
| SVENDSON, STEPHEN | CHAIR |

注意：哈希连接处理过程为对一张数据表以连接列为哈希键，构造哈希表，另一张表使用连接列进行哈希探测，返回满足连接条件的记录。

达梦数据库信创大赛资料

# 实验 11 子查询

## 实验目的：

当一个查询是另一个查询的条件时，称之为子查询。使用子查询的原因是，如果希望执行某 SQL 语句，但是该 SQL 语句还需要依赖于另外一个 SELECT 语句的执行结果。

**语法：**

```
SELECT SELECT_LIST
FROM TABLE
WHERE EXPR OPERATOR
    (SELECT SELECT_LIST FROM TABLE);
```

## 实验步骤：

**例：**

“SALES”表的原数据

| NAME | SALES | DEPTNO |
|---|---|---|
| HANSEN | 1500 | 10 |
| NORDMANN | 800 | 20 |
| PETTERSEN | 1200 | 30 |
| SVENDSON | 3000 | 40 |
| SCOTT | 2500 | 20 |

单行子查询，单行子查询可以使用>,<,<>,=等，单行子查询，子查询的结果唯一非空。

找出和 SCOTT 在同一部门的：

```
SELECT NAME, DEPTNO
FROM SALES
```



达梦信创大赛资料

```
WHERE DEPTNO=(SELECT DEPTNO FROM SALES WHERE NAME=‘SCOTT’);
```

返回结果：

| NAME | DEPTNO |
|---|---|
| NORDMANN | 20 |
| SCOTT | 20 |

注意：上述语句我们最好加上 AND 条件把 SCOTT 用户排除。

# 实验 12 DML 语句

## 实验目的：

能够向现有的表中录入数据，更新数据和删除数据。

## 实验步骤：

INSERT INTO

**用途：**

在表中插入新行。

**语法：**

插入一行数据

```
INSERT INTO TABLE_NAME
VALUES (VALUE1, VALUE2,....)
```

插入一行数据在指定的字段上

```
INSERT INTO TABLE_NAME (COLUMN1, COLUMN2,...)
VALUES (VALUE1, VALUE2,....)
```

**例：**

“PERSONS”表中的原始数据:



达梦数据库信创大赛资料

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| PETTERSEN | KARI | STORGT 20 | STAVANGER |

运行下面的 SQL 插入一行数据：

```
INSERT INTO PERSONS
VALUES ('HETLAND', 'CAMILLA', 'HAGABAKKA 24', 'SANDNES');
```

插入后“PERSONS”表中的数据为：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| PETTERSEN | KARI | STORGT 20 | STAVANGER |
| HETLAND | CAMILLA | HAGABAKKA 24 | SANDNES |

运行下面的 SQL 插入一行数据在指定的字段上：

```
INSERT INTO PERSONS (LASTNAME, ADDRESS)
VALUES ('RASMUSSEN', 'STORGT 67');
```

插入后“PERSONS”表中的数据为：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| PETTERSEN | KARI | STORGT 20 | STAVANGER |
| HETLAND | CAMILLA | HAGABAKKA 24 | SANDNES |
| RASMUSSEN | | STORGT 67 | |

**UPDATE**

**用途：**

更新表中原有数据

**语法：**

```
UPDATE TABLE_NAME SET COLUMN_NAME = NEW_VALUE
WHERE COLUMN_NAME = SOME_VALUE;
```

**例：**

“PERSON”表中的原始数据：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
| --- | --- | --- | --- |
| NILSEN | FRED | KIRKEGT 56 | STAVANGER |
| RASMUSSEN | | STORGT 67 | |

运行下面的 SQL 将 PERSON 表中 LASTNAME 字段为“RASMUSSEN”的 FIRSTNAME 更新为“NINA”：

```
UPDATE PERSON SET FIRSTNAME = 'NINA'
WHERE LASTNAME = 'RASMUSSEN';
```

更新后“PERSON”表中的数据为：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
| --- | --- | --- | --- |
| NILSEN | FRED | KIRKEGT 56 | STAVANGER |
| RASMUSSEN | NINA | STORGT 67 | |

同样的，用 UPDATE 语句也可以同时更新多个字段：

```
UPDATE PERSON
SET ADDRESS = 'STIEN 12', CITY = 'STAVANGER'
WHERE LASTNAME = 'RASMUSSEN';
```

更新后“PERSON”表中的数据为：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
| --- | --- | --- | --- |
| NILSEN | FRED | KIRKEGT 56 | STAVANGER |
| RASMUSSEN | NINA | STIEN 12 | STAVANGER |

**DELETE**

**用途：**

删除表中的数据

**语法：**

```
DELETE FROM TABLE_NAME WHERE COLUMN_NAME = SOME_VALUE
```

**例：**

“PERSON”表中的原始数据:

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| NILSEN | FRED | KIRKEGT 56 | STAVANGER |
| RASMUSSEN | NINA | STIEN 12 | STAVANGER |

删除 PERSON 表中 LASTNAME 为“RASMUSSEN”的数据：

```
DELETE FROM PERSON WHERE LASTNAME = 'RASMUSSEN';
```

执行删除语句后“PERSON”表中的数据为：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
|---|---|---|---|
| NILSEN | FRED | KIRKEGT 56 | STAVANGER |

达梦数据库信创大赛设计

# 实验 13 表管理

## 实验目的：

创建表，修改表，删除表。

## 实验步骤：

CREATE TABLE

**用途：**

建立新的资料表。

**语法：**

```
CREATE TABLE TABLE_NAME
(
COLUMN_NAME1 DATA_TYPE,
COLUMN_NAME2 DATA_TYPE,
.......
);
```

**例：**

创建一张叫“PERSON”的表，该表有 4 个字段"LASTNAME"，"FIRSTNAME"，"ADDRESS"，"AGE"：

```
CREATE TABLE PERSON
(
LASTNAME VARCHAR,
FIRSTNAME VARCHAR,
ADDRESS VARCHAR,
AGE INT
);
```

如果想指定字段的最大存储长度，你可以这样：

```
CREATE TABLE PERSON
(
LASTNAME VARCHAR(30),
FIRSTNAME VARCHAR(30),
ADDRESS VARCHAR(120),
AGE INT(3)
);
```

下表中列出了在 SQL 的一些数据类型：

| DATA TYPE | DESCRIPTION |
|---|---|
| INTEGER(SIZE)<br>INT(SIZE)<br>SMALLINT(SIZE)<br>TINYINT(SIZE) | HOLD INTEGERS ONLY. THE MAXIMUM NUMBER OF DIGITS ARE SPECIFIED IN PARENTHESIS. |
| DECIMAL(SIZE,D)<br>NUMERIC(SIZE,D) | HOLD NUMBERS WITH FRACTIONS. THE MAXIMUM NUMBER OF DIGITS ARE SPECIFIED IN "SIZE". THE MAXIMUM NUMBER OF DIGITS TO THE RIGHT OF THE DECIMAL IS SPECIFIED IN "D". |
| CHAR(SIZE) | HOLDS A FIXED LENGTH STRING (CAN CONTAIN LETTERS, NUMBERS, AND SPECIAL CHARACTERS). THE FIXED SIZE IS SPECIFIED IN PARENTHESIS. |
| VARCHAR(SIZE) | HOLDS A VARIABLE LENGTH STRING (CAN CONTAIN LETTERS, NUMBERS, AND SPECIAL CHARACTERS). THE MAXIMUM SIZE IS SPECIFIED IN PARENTHESIS. |

| | |
|---|---|
| DATE(YYYYMMDD) | HOLDS A DATE |

创建 HUGE 表 ORDERS，表的区大小为 65536 行，文件大小为 64M，指定所在的表空间为 HTS_NAME，做完整镜像，o_comment 列指定的区大小为不做统计信息，其它列（默认）都做统计信息，指定列 o_comment 列按照最大压缩比压缩。

创建 HUGE 表空间 HTS_NAME

create huge tablespace HTS_NAME path '/dm7/data/DAMENG/HTS_NAME';

创建 HUGE 表

CREATE HUGE TABLE ORDERS ( o_orderkey INT, o_custkey INT, o_orderstatus CHAR(1), o_totalprice FLOAT, o_orderdate DATE, o_orderpriority CHAR(15), o_clerk CHAR(15), o_shippriority INT, o_comment VARCHAR(79) STORAGE(stat none) )STORAGE(section(65536) , filesize(64), on HTS_NAME) COMPRESS LEVEL 1 (o_comment) log all;

注意：可以利用 SP_TABLEDEF 过程，查看表的定义，HFS 表像行存储表一样，也可以做增删改查操作，操作方式是完全一样的，但删除及更新操作效率可能比行存储低一些，所以在 HFS 中不宜做频繁的删除及更新操作。因此，HFS 比较适合做分析型表的存储。

当处理复杂的查询或事务时，由于在数据写入永久表之前需要暂时存储一些行信息或需要保存查询的中间结果，可能需要一些表来临时存储这些数据。DM 允许创建临时表来保存会话甚至事务中的数据。在会话或事务结束时，这些表上的数据将会被自动清除。

例：创建基于事务的临时表 tx_temp_tab

CREATE GLOBAL TEMPORARY TABLE tx_temp_tab( EMPNO INT PRIMARY KEY,

ENAME VARCHAR(15) NOT NULL, JOB VARCHAR(10)) ON COMMIT DELETE ROWS;

例：创建基于会话的临时表 sess_temp_tab.

CREATE GLOBAL TEMPORARY TABLE TMP_EMP( EMPNO INT PRIMARY KEY,

ENAME VARCHAR(15) NOT NULL, JOB VARCHAR(10))

ON COMMIT PRESERVE ROWS;

**ALTER TABLE**

**用途：**

在已经存在的表中增加后者移除字段

**语法：**

```
ALTER TABLE TABLE_NAME

ADD COLUMN_NAME DATATYPE;

ALTER TABLE TABLE_NAME

DROP COLUMN COLUMN_NAME;
```

**注意：某些数据库管理系统不允许移除表中的字段**

**例：**

“PERSON”表中的原始数据:

| LASTNAME | FIRSTNAME | ADDRESS |
| --- | --- | --- |
| PETTERSEN | KARI | STORGT 20 |

在 PERSON 表中增加一个名为 CITY 的字段：

```
ALTER TABLE PERSON ADD CITY VARCHAR(30);
```

增加后表中数据如下：

| LASTNAME | FIRSTNAME | ADDRESS | CITY |
| --- | --- | --- | --- |

| | | | |
|---|---|---|---|
| PETTERSEN | KARI | STORGT 20 | |

移除 PERSON 表中原有的 ADDRESS 字段：

```
ALTER TABLE PERSON DROP COLUMN ADDRESS;
```

移除后表中数据如下：

| LASTNAME | FIRSTNAME | CITY |
|---|---|---|
| PETTERSEN | KARI | |

DROP TABLE

**用途：**

在数据库中移除一个数据表定义及该数据表中的所有资料、索引、触发程序、条件约束及权限指定。

**语法：**

```
DROP TABLE TABLE_NAME;
```

达梦数据库信创大赛资料

# 实验 14 约束管理

## 实验目的：

保证数据的完整性和一致性，为表增加约束和删除表上的约束。

## 实验步骤：

**例：**

“EMP”表的表结构如下，且 EMP 表为设置为何约束条件

| 列名 | 类型 | 长度 | 标度 | 是否可以为空 |
|---|---|---|---|---|
| EMPNO | NUMBER | 4 | 0 | Y |
| ENAME | VARCHAR | 10 | 0 | Y |
| JOB | VARCHAR | 9 | 0 | Y |
| MGR | NUMBER | 4 | 0 | Y |
| HIREDATE | DATE | 10 | 0 | Y |
| SAL | NUMBER | 7 | 2 | Y |
| COMM | NUMBER | 7 | 2 | Y |
| DEPTNO | NUMBER | 2 | 0 | Y |

“DEPT”表的表结构如下，且 DEPT 表 DEPTNO 是该表的主键

| 列名 | 类型 | 长度 | 标度 | 是否可以为空 |
|---|---|---|---|---|
| DEPTNO | NUMBER | 2 | 0 | N |
| DNAME | VARCHAR | 14 | 0 | Y |
| LOC | VARCHAR | 13 | 0 | Y |

**语法：**

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT_NAME CONTRIANT_TYPE (COLUM_NAME);
```

**非空约束：**

非空约束是比较特殊的约束，只能做为列级约束，不能做为表级约束。

为“EMP”表的”ENAME”增加非空约束，当然我们也可以在创建表时，指定约束条件。

```
ALTER TABLE EMP MODIFY ENAME NOT NULL;
```

注意：如果表已经有数据，并且该列有 NULL 值，非空的约束我们是无法添加的。

**唯一约束：**

唯一约束，是保证该列值唯一，不能重复。但是 DM7 在录入数据时，会忽略 NULL。唯一约束可以设置成表级和列级。

为”EMP”表的”EMPNO”增加唯一约束。

```
ALTER TABLE EMP ADD CONSTRAINT EMP_UNIQ UNIQUE (EMPNO);
```

注意：我们创建完成，可以通过 USER_CONSTRAINTS 查看。

```
SELECT CONSTRAINT_NAME,TABLE_NAME FROM USER_CONSTRAINTS WHERE
TABLE_NAME='EMP';
```

返回结果：

```
行号       CONSTRAINT_NAME TABLE_NAME
---------- --------------- ----------
1          EMP_UNIQ        EMP
```

**主键约束：**

主键约束一个表只能由一个，可以由一列和多列构成，主键约束的特点唯一非空。主键约束也可以是表级也可以是列级。

为”EMP”表的”EMPNO”增加主键约束。

```
ALTER TABLE EMP ADD CONSTRAINT EMP_PRI PRIMARY KEY (EMPNO);
```

注意：我们创建完成，可以通过 USER_CONSTRAINTS 查看。

```
SELECT CONSTRAINT_NAME,TABLE_NAME FROM USER_CONSTRAINTS WHERE
TABLE_NAME='EMP';
```

返回结果：

```
行号        CONSTRAINT_NAME TABLE_NAME
---------- --------------- ----------
1          EMP_PRI         EMP
```

**外键约束：**

外键约束，一个表可以有多个外键，外键是另外一张表的主键。我们也可以在创建表时指定。

为“EMP”表在“DEPTNO”增加外键约束。

```
ALTER TABLE EMP ADD CONSTRAINT EMP_FK FOREIGN KEY (DEPTNO) REFERENCES
DEPT(DEPTNO);
```

注意：我们创建完成，可以通过 USER_CONSTRAINTS 查看。

```
SELECT CONSTRAINT_NAME,TABLE_NAME FROM USER_CONSTRAINTS WHERE
TABLE_NAME='EMP';
```

返回结果：

```
行号        CONSTRAINT_NAME TABLE_NAME
---------- --------------- ----------
1          EMP_FK          EMP
```

检查约束：

检查约束，是检查该列的值是否符合录入要求。

为“EMP”表在“SAL”增加检查约束

```
ALTER TABLE EMP ADD CONSTRAINT EMP_CHK CHECK (SAL>=800);
```

注意：我们创建完成，可以通过 USER_CONSTRAINTS 查看。

```
SELECT CONSTRAINT_NAME,TABLE_NAME FROM USER_CONSTRAINTS WHERE
TABLE_NAME=’EMP’;
```

返回结果：

```
行号        CONSTRAINT_NAME TABLE_NAME
---------- --------------- ----------
1          EMP_CHK         EMP
```

约束启用和禁用

**语法：**

ALTER TABLE TABLE_NAME ENABLE|DISABLE CONSTRAINT CONSTRAINT_NAME;

禁用“EMP”表的主键约束

```
ALTER TABLE EMP DISABLE CONSTRAINT EMP_PRI;
```

启用“EMP”表的主键约束

```
ALTER TABLE EMP ENABLE CONSTRAINT EMP_PRI;
```

注意：如果在关闭约束的时候，可录入不符合约束条件的数据。但想在启用该约束，就会出错。

**删除约束：**

**语法：**

ALTER TABLE TABLE_NAME DROP CONSTRAINT  CONSTRAINT_NAME;

删除“EMP”的检查约束

```
ALTER TABLE EMP DROP CONSTRAINT EMP_CHK;
```

```
SELECT CONSTRAINT_NAME,TABLE_NAME FROM USER_CONSTRAINTS WHERE
TABLE_NAME=’EMP’;
```

返回结果：

未选定行

达梦数据库信创大赛资料

# 实验 15 视图管理

## 实验目的：

通过视图以不同形式来显示基表中的数据，通过视图可以设定允许用户访问的列和数据行，从而为表提供了额外的安全控制。

**语法：**

```
CREATE [OR REPALCE] VIEW
[<模式名>.]<视图名>[(<列名> {,<列名>})]
AS <表查询> | <表连接>
[WITH CHECK OPTION];
```

## 实验步骤：

视图其实就是一条查询 SQL 语句，用于显示一个或多个表或其他视图中的相关数据。视图将一个查询的结果作为一个表来使用，因此视图可以被看作是存储的查询或一个虚拟表。与表不同，视图不会要求分配存储空间（物化视图例外），视图中也不会包含实际的数据。

**例：**

“EMP”表的原数据，在 SYSDBA 模式下

| EMPNO | ENAME | MGR | SAL | DEPTNO |
|---|---|---|---|---|
| 7369 | SMITH | 7902 | 800 | 20 |
| 7499 | ALLEN | 7698 | 1600 | 30 |
| 7521 | WARD | 7698 | 1250 | 30 |
| 7566 | JONES | 7839 | 2975 | 20 |
| 7654 | MARTIN | 7698 | 1250 | 30 |
| 7698 | BLAKE | 7839 | 2850 | 30 |

| | | | | |
|---|---|---|---|---|
| 7782 | CLARK | 7839 | 2450 | 10 |
| 7839 | KING | <NULL> | 5000 | 10 |
| 7844 | TURNER | 7698 | 1500 | 30 |
| 7900 | JAMES | 7698 | 950 | 30 |
| 7902 | FORD | 7566 | 3000 | 20 |
| 7934 | MILLER | 7782 | 1300 | 10 |

创建视图“VIEW1”，可以查询“EMP”表的“ENAME”，“SAL”

```
CREATE VIEW VIEW1 AS SELECT ENAME,SAL FROM EMP;
```

查看视图“VIEW1”

```
SELECT * FROM VIEW1;
```

结果如下：

| ENAME | SAL |
|---|---|
| SMITH | 800 |
| ALLEN | 1600 |
| WARD | 1250 |
| JONES | 2975 |
| MARTIN | 1250 |
| BLAKE | 2850 |
| CLARK | 2450 |
| KING | 5000 |
| TURNER | 1500 |
| JAMES | 950 |
| FORD | 3000 |
| MILLER | 1300 |

注意：可以利用 USER_VIEWS,查看视图的详细信息。

修改视图“VIEW1”：

```
CREATE OR REPLACE VIEW VIEW1 AS SELECT EMPNO,ENAME,SAL FROM EMP;
```

查看视图“VIEW1”

```
SELECT * FROM VIEW1;
```

结果如下：

| EMPNO | ENAME | SAL |
|---|---|---|
| 7369 | SMITH | 800 |
| 7499 | ALLEN | 1600 |
| 7521 | WARD | 1250 |
| 7566 | JONES | 2975 |
| 7654 | MARTIN | 1250 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7839 | KING | 5000 |
| 7844 | TURNER | 1500 |
| 7900 | JAMES | 950 |
| 7902 | FORD | 3000 |
| 7934 | MILLER | 1300 |

删除视图“VIEW1”：

```
DROP VIEW VIEW1;
```

视图上的 DML 操作：

DML 操作应遵循的原则：

1.简单视图可以执行 DML 操作;

2.在视图包含 GROUP 函数，GROUP BY 子句，DISTINCT 关键字时不能删除数据行;

3.在视图不出现下列情况时可通过视图修改基表数据或插入数据:

A.视图中包含 GROUP 函数，GROUP BY 子句，DISTINCT 关键字;

B.使用表达式定义的列;

C.ROWNUM 伪列。

D.基表中未在视图中选择的其他列定义为非空且无默认值。

视图可用于保持数据库的完整性，但作用有限。

通过视图执行引用完整性约束可在数据库级执行约束。

达梦数据库信创设计

# 实验 16 索引管理

## 实验目的：

在数据库设计中常见的创建三种索引：唯一索引、聚集索引和位图索引，索引的优点：使用索引，加速查询速度，可以降低 I/O 次数，提高数据访问性能。索引的缺点：索引越多，DML 操作的速度越慢，极大的影响 INSERT 和 DELETE。

**语法：**

```
CREATE [OR REPLACE] [CLUSTER][UNIQUE | BITMAP] INDEX <索引名> ON [<模式名>.]<表名>(<索引列定义>{,<索引列定义>});
```

## 实验步骤：

**例：**

“EMP”表的原数据，在 SYSDBA 模式下

| EMPNO | ENAME | MGR | SAL | DEPTNO |
|---|---|---|---|---|
| 7369 | SMITH | 7902 | 800 | 20 |
| 7499 | ALLEN | 7698 | 1600 | 30 |
| 7521 | WARD | 7698 | 1250 | 30 |
| 7566 | JONES | 7839 | 2975 | 20 |
| 7654 | MARTIN | 7698 | 1250 | 30 |
| 7698 | BLAKE | 7839 | 2850 | 30 |
| 7782 | CLARK | 7839 | 2450 | 10 |
| 7839 | KING | <NULL> | 5000 | 10 |
| 7844 | TURNER | 7698 | 1500 | 30 |
| 7900 | JAMES | 7698 | 950 | 30 |

| | | | | |
|---|---|---|---|---|
| 7902 | FORD | 7566 | 3000 | 20 |
| 7934 | MILLER | 7782 | 1300 | 10 |

创建聚簇索引，当建表语句未指定聚集索引键，DM 的默认聚集索引键是 ROWID，即记录默认以 ROWID 在页面中排序。ROWID 是 B 树为记录生成的逻辑递增序号，表上不同记录的 ROWID 是不一样的，并且最新插入的记录 ROWID 最大。

为“EMP”在 EMPNO 列创建聚簇索引：

```
CREATE CLUSTER INDEX EMP_CLUS ON EMP(EMPNO);
```

查看聚簇索引，利用 USER_INDEXES

```
SELECT TABLE_NAME,INDEX_NAME FROM USER_INDEXES WHERE
TABLE_NAME='EMP';
```

结果如下：

| TABLE_NAME | INDEX_NAME |
|---|---|
| EMP | EMP_CLUS |
| EMP | INDEX33555534 |

创建唯一索引，唯一索引可以保证表上不会有两行数据在键列上具有相同的值。

为“EMP”在“EMPNO”列创建唯一索引：

```
CREATE UNIQUE INDEX EMP_UNIQ ON EMP(EMPNO);
```

查看唯一索引，利用 USER_INDEXES

```
SELECT TABLE_NAME,INDEX_NAME FROM USER_INDEXES WHERE
TABLE_NAME='EMP';
```

结果如下：

| TABLE_NAME | INDEX_NAME |
|---|---|

| | |
|---|---|
| EMP | EMP_UNIQ |
| EMP | INDEX33555534 |

创建位图索引，主要针对含有大量相同值的列而创建。位图索引被广泛引用到数据仓库中，创建方式和普通索引一致，对低基数（不同的值很少）的列创建位图索引，能够有效提高基于该列的查询效率。

为“EMP”在 DEPTNO 列创建位图索引：

```
CREATE BITMAP INDEX EMP_BTM ON EMP(DEPTNO);
```

查看位图索引,利用 USER_INDEXES

```
SELECT TABLE_NAME,INDEX_NAME FROM USER_INDEXES WHERE
TABLE_NAME=’EMP’;
```

结果如下：

| TABLE_NAME | INDEX_NAME |
|---|---|
| EMP | EMP_BTM |
| EMP | INDEX33555534 |

重建索引，可以利用函数 SP_REBUILD_INDEX。SP_REBUILD_INDEX 函数要 SCHEAM_NAME 为索引所在的模式名，INDEX_ID 为索引 ID

获取索引 ID

```
SELECT ID,NAME FROM SYSOBJECTS WHERE TABLE_NAME=’EMP_BTM’;
```

结果如下：

| ID | NAME |
|---|---|
| 33555579 | EMP_BTM |

重建位图索引“EMP_TBM”

```
CALL SP_REBUILD_INDEX（‘SYSDBA’, ‘33555579’）;
```

删除索引，索引删除之后，该索引的段的所有簇都返回给包含它的表空间。

删除位图索引“EMP_TBM”

```
DROP INDEX EMP_BTM;
```

达梦数据库信创大赛资料

# 实验 17 数据库开归档

## 实验目的：

掌握 DM8 数据如何开归档及数据文件和日志文件的管理和使用。

## 实验要求：

数据文件以 dbf 为扩展名，它是数据库中最重要的文件类型，一个 DM 数据文件对应磁盘上的一个物理文件，数据文件是真实数据存储的地方，每个数据库至少有一个与之相关的数据文件。在实际应用中，通常有多个数据文件。

重做日志文件以 log 为扩展名。无论何时，在 DM 数据库中添加、删除、修改对象，或者改变数据，都会将重做日志写入到当前的重做日志文件。

## 1、命令方式开归档

例 1：命令方式开归档

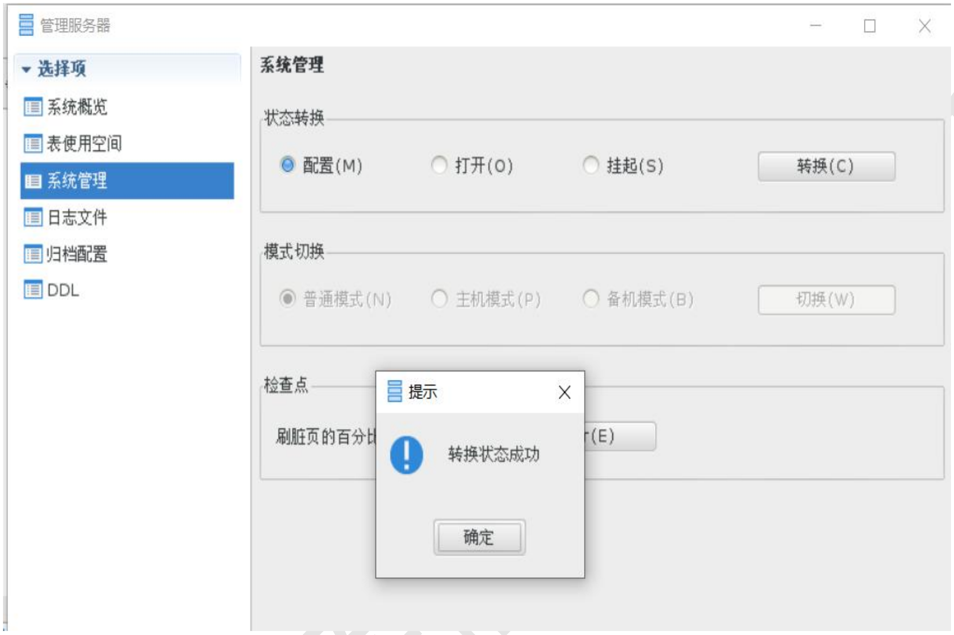
```
SQL> alter database mount;
SQL> alter database add archivelog 'type=local,dest=/dm8/arch,file_size=64,space_limit=0';
SQL> alter database  archivelog;
SQL> alter database open;
SQL> select name,status$,arch_mode from v$database;
行号      NAME   STATUS$     ARCH_MODE
---------- ------ ----------- ---------
1          DAMENG  4           Y
```



达梦数据库信创大赛资料

# 2、利用管理工具开归档

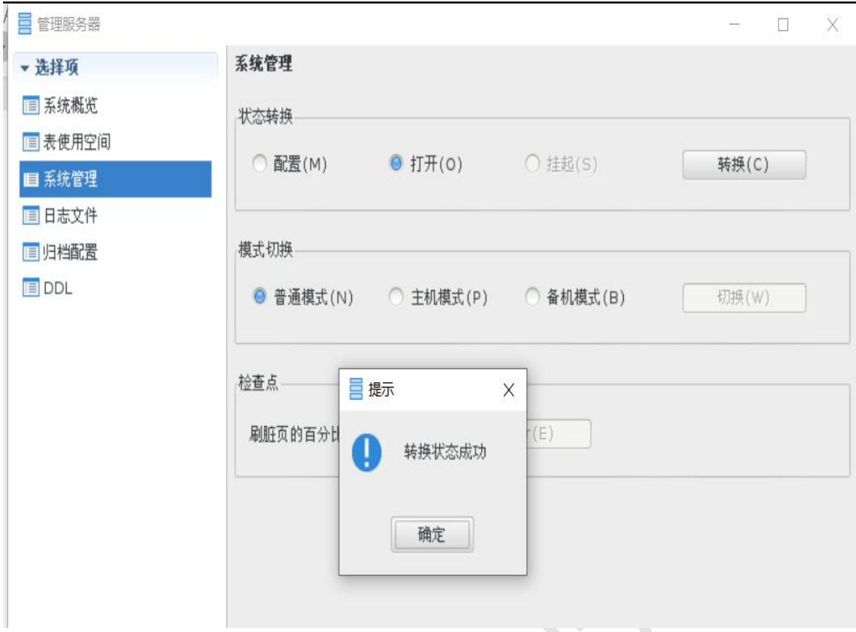
## 步骤 1：管理服务器---系统管理---配置---转换



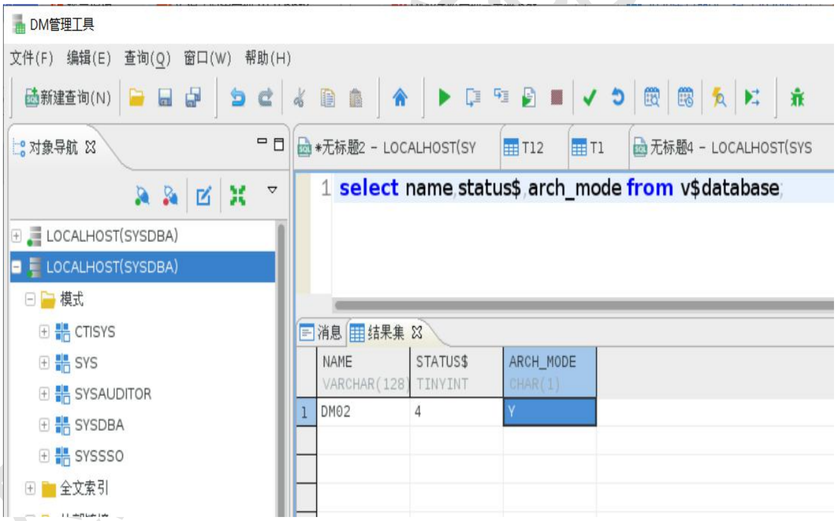

步骤 2：配置归档文件属性




步骤 3：打开数据库


管理服务器
选择项
系统概览
表使用空间
系统管理
日志文件
归档配置
DDL
系统管理
状态转换
配置(M)
打开(O)
挂起(S)
转换(C)
模式切换
普通模式(N)
主机模式(P)
备机模式(B)
切换(W)
检查点
刷脏页的百分比
提示
转换状态成功
确定

步骤 4：查看归档状态


DM管理工具
文件(F) 编辑(E) 查询(Q) 窗口(W) 帮助(H)
新建查询(N)
对象导航
*无标题2 - LOCALHOST(SY
T12
T1
无标题4 - LOCALHOST(SYS
select name,status$,arch_mode from v$database
LOCALHOST(SYSDBA)
LOCALHOST(SYSDBA)
模式
CTISYS
SYS
SYSAUDITOR
SYSDBA
SYSSSO
全文索引
消息
结果集
NAME VARCHAR(128)
STATUS$ TINYINT
ARCH_MODE CHAR(1)
DM02
4
Y

# 实验 18 物理备份与还原

## 实验目的：

掌握数据备份还原的操作。

## 实验要求：

数据库的备份与还原是系统容灾的重要方法。在一个生产系统中，数据库往往处于核心的地位。为了保证数据的安全。达梦数据库管理系统支持数据库和表空间的完全备份和增量备份。

## 实验步骤：

脱机备份，是指关闭了数据库后备份所有数据文件和控制文件的方法。当正常关闭了数据库之后，所有数据库文件的当前 SCN 值完全一致，所以关闭后的数据库备份被称为数据库一致性备份或者冷备份。

## 1、数据库冷备

冷备：dmap 服务是打开的，数据库实例是关闭的。

停止当前运行的数据，如图 18.1

```
[root@localhost bin]# ./DmServiceDMSERVER  stop
Stopping DmServiceDMSERVER:                                [  OK  ]
[root@localhost bin]#
```

图 18.1

例 1.利用控制台工具去备（$DM_HOME/tool/console)

步骤 1：打开 DM 控制台，选择备份还原，如图 18.2

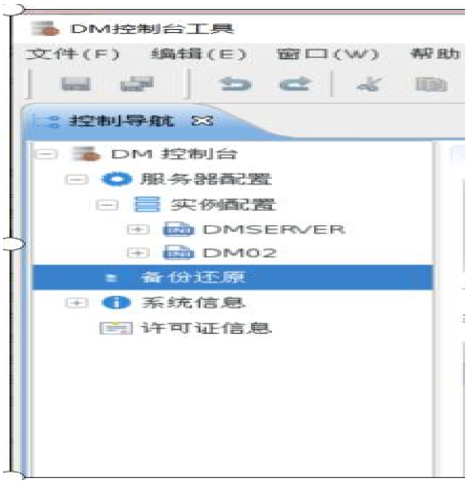



图 18.2

步骤 2：新建备份，如图 18.3




图 18.3

步骤 3：点击确定，如图 18.4

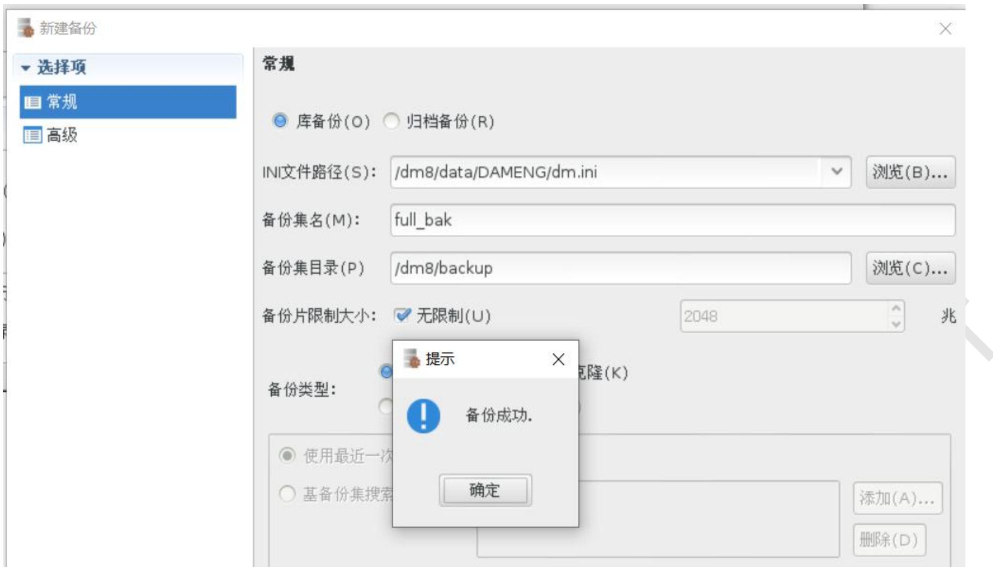
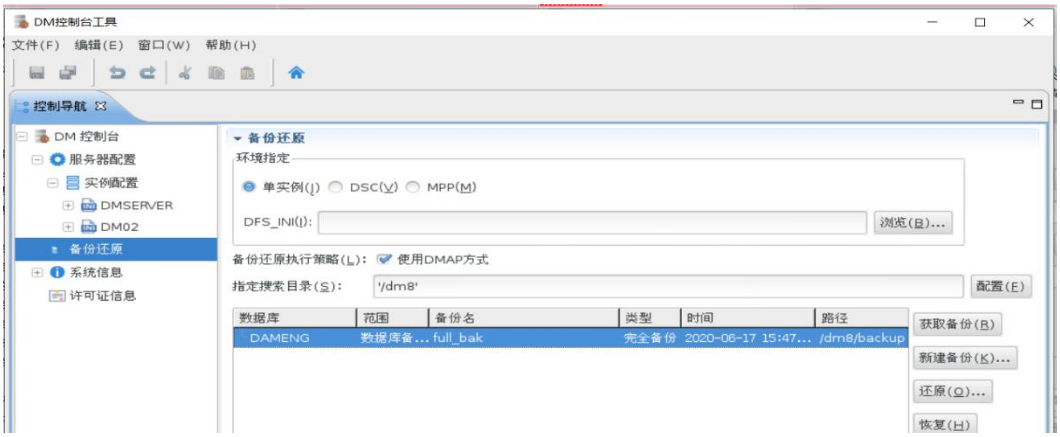



图 18.4

步骤 4：点击确定，备份完成 如图 18.5




如图 18.5

我们也可以使用 DMRMAN 做脱机备份

例 2：利用 DMRMAN 进行脱机备份

步骤 1：停止当前运行的数据，如图 18.6。

```
[root@localhost bin]# ./DmServiceDMSERVER  stop
Stopping DmServiceDMSERVER:                                [  OK  ]
[root@localhost bin]# 
```

图 18.6

步骤 2、利用 DMRMAN 进行脱机备份

[dmdba@localhost bin]$ cd /dm8/bin

[dmdba@localhost bin]$ ./dmrman

dmrman V8

RMAN> backup database '/dm8/data/DAMENG/dm.ini';

```
DBF BACKUP MAIN......
BACKUPSET [/dm8/data/DAMENG/bak/DB_DAMENG_FULL_20200617_155126_640790] END, COD
E [0]......
META GENERATING......
total 13 packages processed...
total 13 packages processed...
total 13 packages processed!
CMD END.CODE:[0]
backup successfully!
time used: 00:00:02.827
RMAN> 
```

## 2、数据库热备

热备 dmap 服务是打开的，数据库实例是打开的，数据库是归档模式。

**例 3：**归档模式下，对数据库做完全备份。

步骤 1：指定工作目录，如图 18.7-1

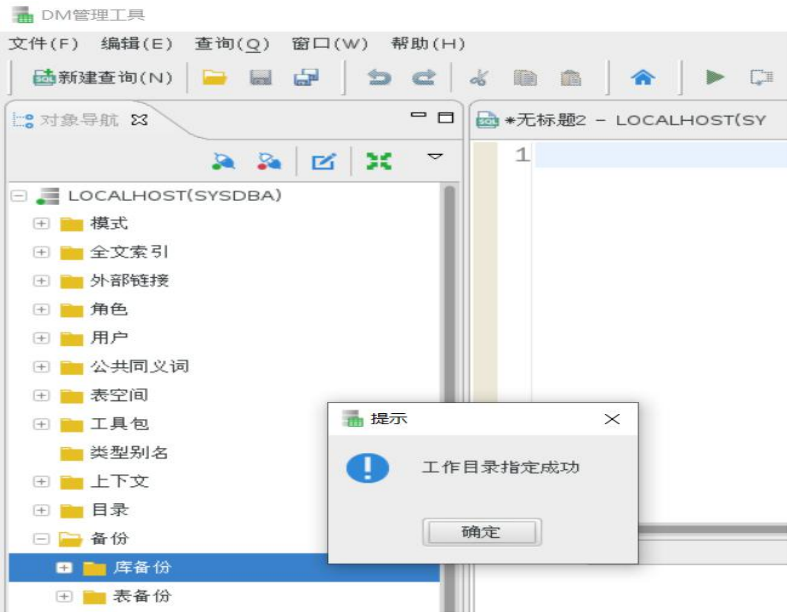
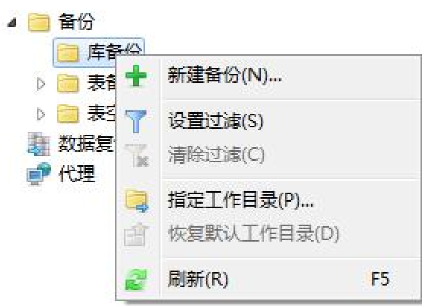



图 18.7-1

步骤 2：选择备份，选择库备份，选择新建备份，如图 18.7-2。




18.7-2

步骤 3：新建库备份，如图 18.8

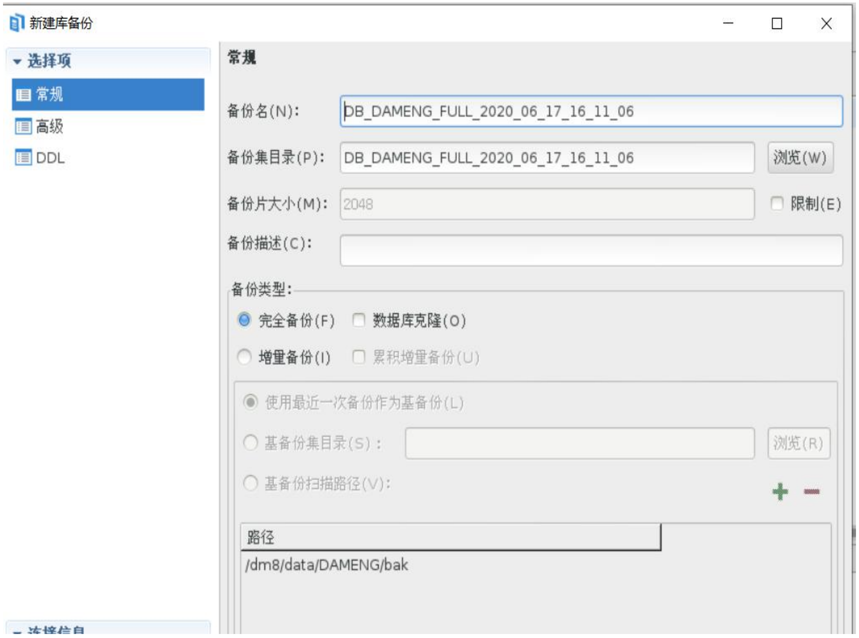



图 18. 8

步骤 4：点击确定，备份完成，查看备份，如图 18.9

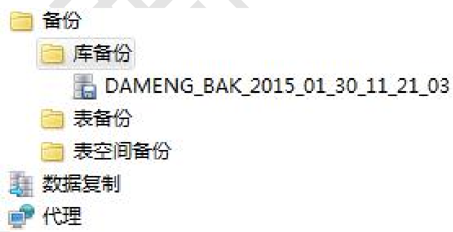



图 18.9

**例** 4：命令行方式下全备，增量备

全备

```
SQL> backup database full backupset '/dm8/backup/fullbak2';
```

增量备

```
SQL> backup database increment backupset '/dm8/backup/incr_bak';
```

# 3、数据库还原

**例 5：**数据还原操作，假设实例 DMSERVER 出现故障，还原到上次备份的时刻。

步骤 1：打开 DM 控制台，选择备份还原，如图 18.10。




图 18.10

步骤 2：选中要恢复的数据库备份，点击“还原”，如图 18.11。




图 18.11

步骤 3：点击“确定”，还原完成，如图 18.12。

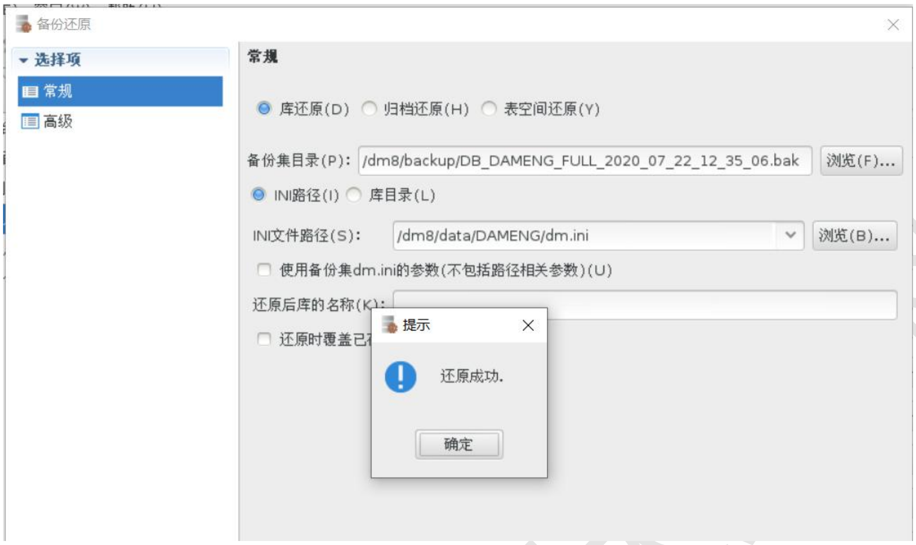



图 18.12

步骤 4：点击恢复，如图 18.13

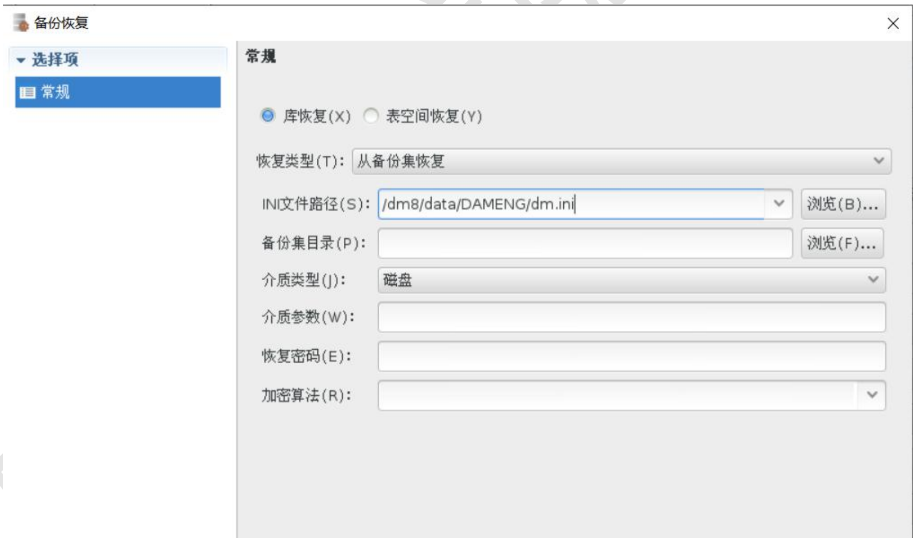



图 18.13

步骤 5：点击确定，恢复成功如图 18.14

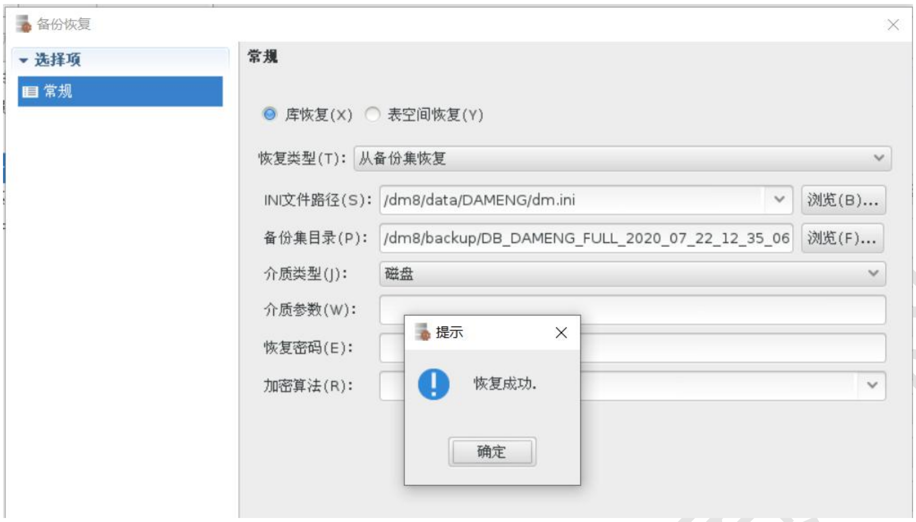



图 18.14

# 实验 19 逻辑备份与还原

## 实验目的：

掌握对数据库的逻辑备份和还原。

## 实验要求：

dexp 工具可以对本地或者远程数据库进行数据库级、模式级和表级的逻辑备份。

## 实验步骤：

**例 1**：对数据库 FULL 导出。

```
./dexp USERID=SYSDBA/SYSDBA@192.168.0.248:8888/user/data
FILE=db_str.dmp DIRECTORY=/mnt/dexp/data LOG=db_str.log FULL=Y
```

在导出的过程中可以 DIRECTORY 参数以指定路径，如果不指定路径，程序将根据当前的运行环境来设置相应的导出路径。

**PWD 密码中带有特殊字符（/,@.:等），需要外加三层双引号（""""""）**

进行转义。

dimp 工具利用 dexp 工具生成的备份文件对数据库进行联机逻辑还原。还原的方式可以灵活选择，如是否忽略对象存在而导致的创建错误、是否导入约束、是否导入索引、导入时是否需要编译、是否生成日志等。

**例 2：**FULL 方式导入数据

./dimp USERID=SYSDBA/SYSDBA@192.168.0.248:8888 FILE=db_str.dmp DIRECTORY=/mnt/data/dexp LOG=db_str.log FULL=Y IGNORE=Y

注意：IGNORE 指明当导入的表存在时，如果其值为 FALSE，则不导入数据，否则导入数据

达梦数据库信创入门资料